

GLOBAL OPTIMIZATION: SOFTWARE AND APPLICATIONS

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Marina Schmidt

©Marina Schmidt, May/2017. All rights reserved.

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science

176 Thorvaldson Building

110 Science Place

University of Saskatchewan

Saskatoon, Saskatchewan

Canada

S7N 5C9

OR

Dean College of Graduate and Postdoctoral Studies University of Saskatchewan

116 110 Science Place

University of Saskatchewan

Saskatoon, Saskatchewan

Canada

S7N 5C9

ABSTRACT

Mathematical models are a gateway into both theoretical and experimental understanding. However, sometimes these models need certain parameters to be established in order to obtain the optimal behaviour or value. This is done by using an optimization method that obtains certain parameters for optimal behaviour, as described by an objective function that may be a minimum (or maximum) result. Global optimization is a branch of optimization that takes a model and determines the global minimum for a given domain. Global optimization can become extremely challenging when the domain yields multiple local minima. Moreover, the complexity of the mathematical model and the consequent lengths of calculations tend to increase the amount of time required for the solver to find the solution. To address these challenges, two software packages were developed to aid a solver in optimizing a black box objective function. The first software package is called Computefarm, a distributed local-resource computing software package that parallelizes the iteration step of a solver by distributing objective function evaluations to idle computers. The second software package is an Optimization Database that is used to monitor the global optimization process by storing information on the objective function evaluation and any extra information on the objective function. The Optimization Database is also used to prevent data from being lost during a failure in the optimization process.

In this thesis, both Computefarm and the Optimization Database are used in the context of two particular applications. The first application is quantum error correction gate design. Quantum computers cannot rely on software to correct errors because of the quantum mechanical properties that allow non-deterministic behaviour in the quantum bit. This means the quantum bits can change states between $(0, 1)$ at any point in time. There are various ways to stabilize the quantum bits; however, errors in the system of quantum bits and the system to measure the states can occur. Therefore, error correction gates are designed to correct for these different types of errors to ensure a high fidelity in the overall circuit. A simulation of a quantum error correction gate is used to determine the properties of components needed to correct for errors in the circuit of the qubit system. The gate designs for the three-qubit and four-qubit systems are obtained by solving a feasibility problem for the intrinsic fidelity

(error-correction percentage) to be above the prescribed 99.99% threshold. The Optimization Database is used with the *MATLAB*'s Global Search algorithm to obtain the results for the three-qubit and four-qubit systems. The approach used in this thesis yields a faster high-fidelity ($\leq 99.99\%$) three-qubit gate time than obtained previously, and obtained a solution for a fast high-fidelity four-qubit gate time. The second application is Rational Design of Materials, in which global optimization is used to find stable crystal structures of chemical compositions. To predict crystal structures, the enthalpy that determines the stability of the structure is minimized. The Optimization Database is used to store information on the obtained structure that is later used for identification of the crystal structure and Compute-farm is used to speed up the global optimization process. Ten crystal structures for carbon and five crystal structures for silicon-dioxide are obtained by using Global Convergence Particle Swarm Optimization. The stable structures, graphite (carbon) and cristobalite (silicon dioxide), are obtained by using Global Convergence Particle Swarm Optimization. Achieving these results allows for further research on the stable and meta-stable crystal structures to understand various properties like hardness and thermal conductivity.

ACKNOWLEDGEMENTS

Thank you to my supervisor, Dr. Raymond Spiteri, for giving me an opportunity to do my masters in the Numerical Simulations Laboratory. He shared his passion for solving science problems with my passion for physics and computer science. His guidance, patience and drive helped me see my fullest potential in research and his never ending patience with my writing skills that slowly got better. I would also like to say thank you to my other supervisor, Dr. John Tse, for getting me involved in a very interesting physics problem and including me with his lab. We enjoyed many conversations over chemistry, computers and FORTRAN. I would like to give my thank you to my parents, and aunts and uncles. My parents, Mary and Myron Schmidt, always trying to understand my research and being proud of me. My aunt, Donna Vroom for being a role model for always achieve a higher educational degree. My aunt, Linda Merrithew, and uncle, Karl Vroom, for being supportive of me.

I would like to dedicate my thesis to my aunt, Fern Clark, and grandmother, Alice Vroom, for teaching me to always persevere and their endless amount of support.

CONTENTS

Permission to Use	i
Abstract	ii
Acknowledgements	iv
Contents	vi
List of Tables	viii
List of Figures	ix
List of Abbreviations	x
List of Symbols	xi
1 Introduction	1
1.1 Global optimization	2
1.2 Software Packages	3
1.2.1 Computefarm	3
1.2.2 The Optimization Database	4
1.2.3 PythOPT	4
1.3 Contributions	4
1.4 Overview	5
2 Global Optimization	7
2.1 Global Optimization Problems	7
2.2 Objective Function	8
2.3 Constraints	11
2.3.1 Global optimization algorithm constraints	12
2.4 Algorithms	16
2.4.1 Deterministic	16
2.4.2 Stochastic	17
3 Software	25
3.1 Computefarm	26
3.1.1 Motivation	26
3.1.2 Requirements	27
3.1.3 Structure	27
3.1.4 Performance analysis	34
3.2 Optimization Database	35
3.2.1 Motivation	35

3.2.2	Requirements	38
3.2.3	Database schema	39
3.2.4	Monitor schemes	42
4	Applications	47
4.1	Quantum error correction circuit design	47
4.2	Crystal structure prediction	54
4.3	Concluding remarks	59
5	Conclusions and suggestions for future research	63
5.1	Conclusions	63
5.1.1	Overview of the software	63
5.1.2	Quantum error correction gate design	64
5.1.3	Rational Design of Materials	65
5.2	Suggestions for future work	65
5.2.1	Software packages	65
5.2.2	Quantum computer error correction gate design	66
5.2.3	Rational Design of Materials	66
	References	68
	Appendix A Software application interface	72
A.0.1	ComputeFarm interface	72
A.0.2	Performance test specifications	75
	Appendix B Optimization Database	76
B.0.1	Optimization Database interface	76
B.0.2	Email interface	79
B.0.3	Website interface	81
	Appendix C Quantum error correction gate results	82
	Appendix D Rational Design of Materials Files	83

LIST OF TABLES

3.1	ComputeFarm performance comparison to SPSO in pythOPT.	35
4.1	Search space for Rational Design of Materials project for an n -atom system.	58
4.2	Time comparisons between GCPSO and GCPSO with ComputeFarm for silicon dioxide.	59
A.1	Global optimization and computer specifications used for the ComputeFarm performance experiment.	75
C.1	Duration time results for intrinsic fidelity for the three-qubit case.	82
C.2	Duration time results for intrinsic fidelity for the four-qubit case.	82
C.3	Duration time results for intrinsic fidelity for the five-qubit case.	82
D.1	Search space for Rational Design of Materials project for an n -atom system.	83

LIST OF FIGURES

2.1	Comparison between a convex and non-convex function.	10
2.2	Convex segmented regions of a non-convex function.	15
2.3	Schematic of the particle movement using (2.7).	21
3.1	Process of a global optimization algorithm using Computefarm.	28
3.2	Process of a global optimization algorithm using Computefarm to evaluate positions.	29
3.3	Computefarm connection handler work flow.	33
3.4	The workflow of monitoring lost client connections.	34
3.5	Optimization Database schema. One settings table to many problem tables.	41
3.6	Snapshot of a periodic email notification displaying the information stored in the Optimization Database for the 4-qubit problem.	43
3.7	Inactive email notification.	44
3.8	Completion email notification.	45
3.9	Snapshot of the information displayed information on multiple 4-qubit instance monitored by the Optimization Database.	46
4.1	The possible spin states of a qubit.	48
4.2	Chain of four qubits in a 4-qubit system.	50
4.3	The interaction strength between the first transmon ($k = 0$) and other transmons is shown by the thickness of the lines connecting them.	52
4.4	Piecewise pulse for the three-qubit case with a duration gate time of 23 ns.	54
4.5	Piecewise pulse for the four-qubit case with a duration gate time of 70 ns.	54
4.6	An unit cell showing the unit lengths ($\mathbf{a}, \mathbf{b}, \mathbf{c}$), the angles (α, β, γ), and the reciprocal lattice basis vectors ($\mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^*$).	56
4.7	Lattice mesh based on the KSPACING value.	57
4.8	Ten minimal enthalpy structures of carbon.	60
4.9	Graphite (C 2/m).	60
4.10	Cubic-diamond, (Fd3m).	61
4.11	Five minimal enthalpy structures of silicon dioxide.	61
4.12	Cristobalite ($\bar{I}4_2d$).	62
4.13	Stishovite ($P4_2mm$).	62

LIST OF ABBREVIATIONS

PSO	Particle Swarm Optimization
SPSO	Standard PSO
GCPSO	Global Convergence PSO
BOINC	Berkeley Open Infrastructure for Network Computing
CALYPSO	Crystal structure AnaLYsis by PSO

LIST OF SYMBOLS

f	objective function
\mathbb{D}	search space
\mathbf{x}	decision vector
\mathbf{x}^*	optimal decision vector
$\mathbf{g}_=$	equality constraint function
\mathbf{g}_\leq	inequality constraint function
\mathbb{F}	feasible set
\mathbf{U}	Unitary matrix
\mathbf{H}	Hamiltonian matrix
\mathbf{A}^\dagger	complex conjugate transpose of \mathbf{A}

CHAPTER 1

INTRODUCTION

Optimization is a process to obtain a minimal (or maximal) result for a defined problem. Problems can range from optimization of simple objective functions, for example $f(x) = x^2$, to complex problems that contain multiple equations with various properties. By optimizing the objective functions, parameters for a given objective function are found. This can lead to new discoveries in research and further the development of new technologies.

Optimization can be local or global. Local optimizations search in a given neighbourhood of a provided candidate solution. When searching for a minimum in a case where the local minimum is not guaranteed to be the global minimum, a global solver is needed. Unlike local optimization, global optimization searches the whole domain (rather than a particular neighbourhood) for the optimum. Global optimization algorithms have found global solutions for applications including chemical equilibrium, nuclear reactors, curve fitting, vehicle design and cost, and many others [31].

Global optimization can not always guarantee to obtain a global solution for a given set of computational resources; thus, approximations are necessary. Certain global optimization algorithms cater to specific problems, and this has led to a diversity of developed global optimization algorithms. Typically, the focus of a global optimization algorithm is to obtain an acceptable result in an acceptable amount of computational time; however, developing a new method to solve a specific problem is challenging, and other methods may be shown to be better.

The first step is selecting a global optimization algorithm that suits the problem properties. Various types of properties of the problem can cause challenges for the global optimization process, for example the evaluation time of the objective function evaluation can prolong the overall optimization time more than desired. Another consideration when se-

lecting an algorithm is the type of constraints imposed on the problem. This can narrow down the choice to a subset of algorithms that can be used to solve the problem. In some cases, additional software is used to assist the global optimization process. An example of assistance is parallelizing the code using various software libraries or techniques. By parallelizing the optimization process for long objective function evaluations the overall iteration time to evaluate a number of objective functions decreases. Various algorithms offer different parallel versions [36]. One method to parallelize a global optimization algorithm is to use a distributed system to reduce the number of resources needed on one machine and the speed up the overall optimization time. Some examples of distributed global optimization methods are MapReduce with Particle Swarm Optimization-Genetic Algorithm [32] and BOINC with PSO and Differential Evolution [13].

In this thesis, two software packages are developed to assist the global optimization process. One software package is Computefarm, a distributed local-resource computing program that utilizes the client-server model to distribute objective function evaluations to multiple client machines. By distributing objective function evaluations to multiple client machines, multiple evaluations can be done simultaneously. This distribution reduces the time spent on the iteration step of the global optimization algorithm and reduces the computer resources needed for one machine. The other software package is the Optimization Database, a relational database to monitor the progress of global optimizations that has built-in features to store extra information on the application and notifies the users through email about the progress of solving the application. Both software packages are used in solving the applications: quantum error correction gate design and Rational Design of Materials.

1.1 Global optimization

Global optimization is the process of finding the global minimum value of a function. The function is referred to as the *objective function*. To solve a global optimization problem is to find \mathbf{x}^* such that

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{D}} f(\mathbf{x}),$$

where \mathbb{D} is the domain of the objective function f . Vector \mathbf{x}^* represents the solution to the problem. How a global optimization solver obtains \mathbf{x}^* is how methods differ. There are two main categories of global optimization methods: deterministic and stochastic.

Deterministic algorithms are rigorous algorithms that use no randomness for selecting a potential solution \mathbf{x}^* . Given enough time to solve the problem, deterministic methods typically converge to a global minimum value. However, their rigorous conditions can become computationally excessive for high-dimensional models or complicated functions.

Stochastic algorithms are algorithms that utilize randomness in an adaptive search for a global solution. Typically, stochastic algorithms obtain a good candidate solution for high dimensional problems in a relatively shorter amount of time than deterministic algorithms. However, these methods sacrifice the possibility of a guaranteed global solution within a finite number of computations. Some examples of such algorithms are Particle Swarm Optimization [21], Genetic Algorithm [4], and Simulated Annealing [4]. Two algorithms focused on in this thesis are Global Convergence Particle Swarm Optimization and Global search, both discussed in Chapter 3.

1.2 Software Packages

Two software packages are developed to assist the global optimization solvers in obtaining results for the two applications in this thesis. The software packages are: Computefarm, and the Optimization Database. One other software package used in combination with Computefarm is pyhOPT [42], a problem solving environment.

1.2.1 Computefarm

Computefarm is a distributed local-resource computing system that uses idle computer resources on the various client (farmed) computers to run multiple objective function evaluations at once. It can speed up the iteration process of a given solver thereby speeding up the search for the global minimum. It also has fault tolerance to failures of a farmed computer by allowing the global optimization process to continue without the need to restart. The software package has internal parallelism to allow for multiple connections to open and

facilitate other functionalities like monitoring client machines continuously.

1.2.2 The Optimization Database

The Optimization Database is a flexible database that is used by the objective function to store the results and any extra information pertaining to the application that may be used for post-processing of the data. With this information, the user can i) determine if the solver is stuck at a local minimum, ii) initiate other solvers based on currently stored data, or iii) use the data to further analyse the model. Other features of the software package include an emailing script that notifies the user of the status and current results of various instances of the application and a website that shows the current status of the optimization in real-time. These features allow for convenient monitoring of the progress on the global optimization process.

1.2.3 PythOPT

PythOPT is a problem-solving environment for optimization methods [42]. The open-source software package provides multiple optimization algorithms including various PSO variations. This package is used in solving multiple global optimization problems including the Ration Design of Materials problem (discussed in Chapter 4.2). The open-source package allows for easy integration with Computefarm and the pre-implement global optimization algorithms.

1.3 Contributions

The two main software contributions of this thesis are Computefarm and the Optimization Database; Computefarm helps parallelizes global optimization solvers, and Optimization Database monitors and stores the progress of the global optimization. The main application contributions are quantum error correction gate design and Rational Design of Materials. In the first application, a gate is designed to correct for errors in quantum-processor systems. Unlike transistor-based computers, quantum computers cannot be controlled using a software-based design. Instead, they are controlled directly by a gate component. This makes

the process of manufacturing gates and the testing of desired reliability quite costly. In light of this, several models have been designed to simulate a quantum error correction gate for the purpose of determining the effect of error correction on a given N -qubit system. To ensure high reliability (also known as fidelity) of the gate design, the gate parameters are optimized so that the model returns the desired fidelity of at least 99.99% [6, 14]. This is the minimal modelled fidelity needed for a gate design to achieve an experimental fidelity of 99.9% due to other experimental errors like decoherence. The gate designs for three-qubit and four-qubit systems are solved by reformulating the problem as a feasibility problem. This allows further insight on solving multi-qubit and quantum control problems that can lead to experimental manufacturings of the multi-qubit circuits.

Rational Design of Materials has been used in the past decade to approximate the most stable structures of a chemical composition at a particular temperature and pressure environment. By using global optimization methods, various stable and metastable structures are obtained for further analysis on the structure. In combination with the global optimization algorithm another software known as the Vienna Ab-initio Simulation Package (VASP) [24, 25] is used to determine the stability of the crystal structure. Other software packages like Crystal structure AnaLYsis by PSO (CALYPSO) [43] have been developed to obtain crystal structures because the global optimization method is more cost-effective when compared to experimental testing. In this thesis, carbon and silicon dioxide are globally optimized to find the ten and five most stable structures using Global Convergence Particle Swarm Optimization with Computefarm and Optimization Database. Solving for stable and metastable structures allows for further research into properties of the crystal structures that cannot be obtained experimentally due to cost or because it is not experimentally possible. Some examples of properties of interest are hardness, thermal conductivity, and superconductivity. Exploring these structures theoretically is a more cost-effective method compared experimentally.

1.4 Overview

The remainder of this thesis is structured as follows. Chapter 2 gives the background on the global optimization algorithms. Chapter 3 documents the software developed, Computefarm and the Optimization Database. Chapter 4 describes the two applications, quantum error correction gate design and Rational Design of Materials, as well as how the software was used to aid in the solving of these applications. Finally, Chapter 5 summarizes the results obtained for the two applications and the benefits the software provided for solving the applications.

CHAPTER 2

GLOBAL OPTIMIZATION

Global optimization is a widely used method in engineering, chemistry, and economics, among other fields, to solve optimization applications. In some cases, an application presents little information to solve it or is considered hard to solve due to the objective function's properties. In this situation, a global optimization method is used to find either a global minimum (or maximum) or a feasible solution.

This chapter presents the definition and notation for a global optimization problem, the potential properties of the problem, and relevant global optimization methods to solve the problem. Section 2.1 defines a global optimization problem and what is considered to be a solution. Section 2.2 describes the various properties of an objective function and the challenges and benefits of each of these when globally optimizing. Section 2.3 defines the type of constraints that are potentially present in global optimization. Section 2.4 discusses relevant global optimization algorithms used to solve the applications in Chapter 4.

2.1 Global Optimization Problems

Global optimization is the process of obtaining the extreme minimum (or maximum) value of a function. Two key concepts that define a global optimization problem are the *objective function* $f : \mathbb{D} \rightarrow \mathbb{R}$ and the *search space* \mathbb{D} . An element of the search space is known as the decision vector \mathbf{x} . Let (f, \mathbb{D}) be known as an *unconstrained optimization problem*. Let there exist a vector $\mathbf{x}^* \in \mathbb{D}$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbb{D}.$$

A solution of an optimization problem is a value of $f(\mathbf{x}^*)$, known as a *minimum*,

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{D}} f(\mathbf{x}). \quad (2.1)$$

A maximization problem minimizes $-f(\mathbf{x})$ in (2.1) without loss of generality. In this thesis, only the minimization problem is considered.

Two types of constraints can be added to the global optimization problem: *equality constraints*, $n_{=}$,

$$\mathbf{g}_{=} : \mathbb{D} \rightarrow \mathbb{R}^{n_{=}}, \quad (2.2)$$

and *inequality constraints*, n_{\leq} ,

$$\mathbf{g}_{\leq} : \mathbb{D} \rightarrow \mathbb{R}^{n_{\leq}}. \quad (2.3)$$

Combining constraints (2.2) and (2.3) with (2.1) creates a set of feasible decision vectors \mathbb{F}

$$\mathbb{F} = \{ \mathbf{x} : \mathbf{x} \in \mathbb{D}, \mathbf{g}_{=}(\mathbf{x}) = \mathbf{0}, \mathbf{g}_{\leq}(\mathbf{x}) \leq \mathbf{0} \}.$$

This is known as a *constrained global optimization problem*

$$\min_{\mathbf{x} \in \mathbb{F}} f(\mathbf{x}).$$

A solution in the set \mathbb{F} is a solution to a constrained optimization problem.

2.2 Objective Function

A global optimization problem is defined as finding the decision vector \mathbf{x} in the solution space set \mathbb{D} in which an objective function $f : \mathbb{D} \rightarrow \mathbb{R}$ attains a minimum. A global optimization problem can be classified based on objective functions properties, some examples are:

- continuity
- convexity
- differentiability
- evaluation time

Continuity An objective function is continuous at the decision vector $x = a$ if and only if

$$\lim_{x \rightarrow a} f(x) = f(a);$$

otherwise, it is discontinuous at a . An example of this is a *discrete* objective function where decision vectors \mathbf{x} are only specific decision vectors in the domain, \mathbb{D} (e.g., whole numbers). In global optimization, knowing the continuity influences which algorithm to use for the global optimization process. For discrete objective functions, specific algorithms have been developed to handle the space constraints. Some examples of these algorithms are Discrete Particle Swarm Optimization [20] and the Branch-and-Bound algorithm [28] in which numbers are rounded up to the nearest value. A second example of a discontinuous objective function is when the objective function is not defined at a decision vector \mathbf{x} in \mathbb{D} . Constraints can be used to overcome this challenge, by constraining the \mathbb{D} to not include the regions where the objective function evaluation is not defined (see Section 2.3).

Convexity A function is *convex* if for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{D}$ it follows that,

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2), \quad 0 \leq \lambda \leq 1. \quad (2.4)$$

If (2.4) does not hold, the objective function is *non-convex*. When an objective function is convex, local minima are also global minima; therefore, only a local optimizer is needed to solve the problem. However, when the objective function is non-convex not every local minimum is a global minimum, and the whole domain needs to be searched to attain a global minimum, as shown in Figure 2.1.

Differentiability A function is said to be *differentiable* at a decision vector \mathbf{x}^* in \mathbb{D} if its derivative exists at \mathbf{x}^* . The multi-variable generalization of the derivative is known as the gradient. The direction of a gradient provides information on local minima and maxima in \mathbb{D} . Some optimization solvers exploit this information to converge to a solution faster by using the direction and magnitude of the slope to determine if a minimum is obtained. The first-order conditions for a local minimum state that in an open neighbourhood of \mathbf{x}^*

$$\nabla f(\mathbf{x}^*) = \mathbf{0}.$$

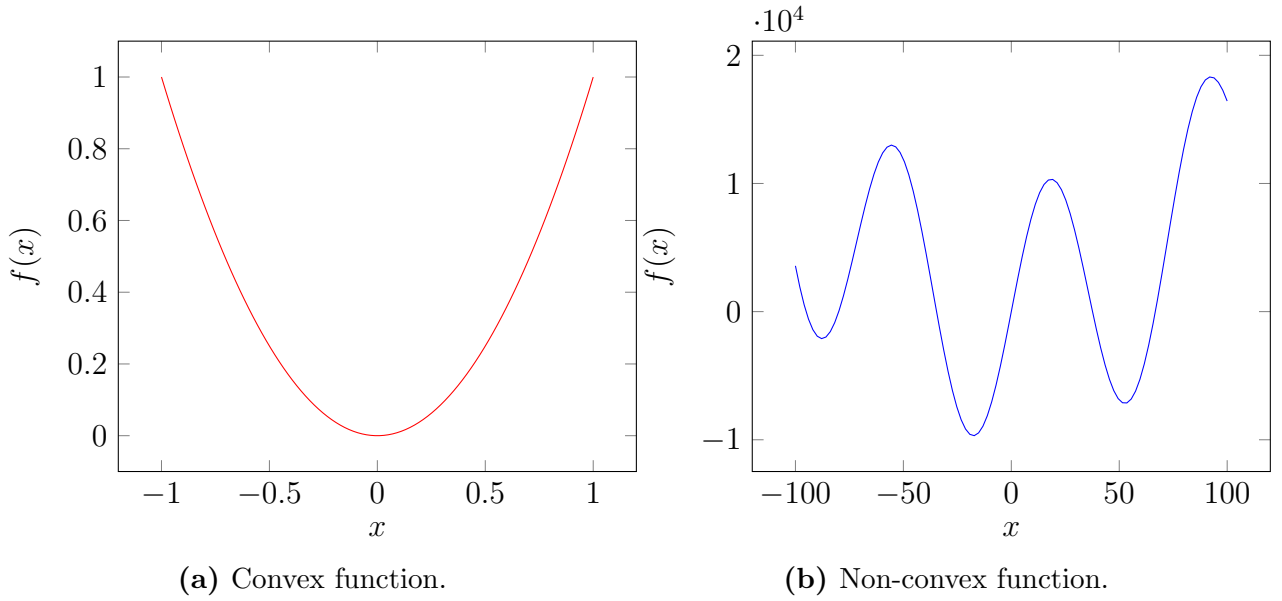


Figure 2.1: Comparison between a convex and non-convex function.

The second-order conditions state that if \mathbf{x}^* is a local minimum of f and $\nabla^2 f$ exists and is continuous in the open neighbourhood \mathbf{x}^* , then $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}^*)$ implies that positive semi-definite. Positive semi-definite is for any $\mathbf{p} \neq \mathbf{0}$,

$$\mathbf{p}^T \nabla^2 f(\mathbf{x}^*) \mathbf{p} \geq 0,$$

then \mathbf{x}^* is a local minimum. Some examples of gradient-based local solvers are Newton's method and gradient-descent.

Some global optimization algorithms use gradient information of the objective function to obtain the local minimum. These methods are known as hybrid methods because they use gradient-based local solvers to attain local minimum, then compare the local minima to obtain a global minimum. An example of a hybrid method is the Multi-Level Single Linkage algorithm [28] that finds local minima at each search decision vector using gradient-based methods to obtain a global minimum.

Commonly objective functions do not provide external information on the gradient of the objective function; therefore, a gradient-free method is used. Methods that do not take in information on the objective function's gradient are also known as *direct search* methods.

These methods are described to have an order relation between two decision vectors [17]. The order relation between two decision vectors is a valued property between the two decision vectors. A common relation is the value of the objective function to determine which one is most likely a minimum; this can be later classified as a global best value for a given iteration step. Other relations can include distances between decision vectors, constraint violations, or if decision vectors are in a given region such as a basin of attraction. An example of a direct search global optimization algorithm is generalized pattern search [39].

Evaluation time *Evaluation time* is the time it takes to evaluate the objective function at decision vector \mathbf{x} . Long evaluation times are computationally expensive and slow down the overall global optimization process. Global optimization methods have been optimized, and sometimes parallelized, to evaluate multiple objective function evaluations simultaneously to speed up the optimization process. An example of this is parallel particle swarm optimization [18] (see Section 2.4.2).

Objective function properties determine if a problem is easy or hard, and that can help determine which algorithm to use. For situations in which an objective function’s properties are unknown or cannot be provided, a *black-box global optimization* solver is used. The term “black-box” refers to an objective function that provides no prior knowledge of the objective function evaluation. It only takes in a decision vector \mathbf{x}^* and returns an objective function value. Examples of black-box global optimization solvers are Genetic Algorithm [4], Differential Evolution [4], and Particle Swarm Optimization [21].

2.3 Constraints

Optimization problems are often constrained global optimization problems because of physical limitations on models. Constraints can also remove the ambiguity of the relation between two decision vectors that result in the same objective function value. If two decision vectors have the same objective function evaluation value, constraints can be used to determine which one is potentially closer to the global solution by assigning a penalty value to the decision vector that does not satisfy the constraints. Constraints can be set in the global

optimization algorithm or set within the objective function.

2.3.1 Global optimization algorithm constraints

Global optimization algorithm constraints are constraints given by an explicit mathematical formula, e.g. set by the domain \mathbb{D} (*bound constraints*), equality (2.2) and inequality constraints (2.3). Bound constraints are simple constant inequality constraints

$$\mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U,$$

where \mathbf{x}_L and \mathbf{x}_U are the lower and upper bounds imposed upon each component of decision vector, $\mathbf{x} \in \mathbb{D}$.

Equality constraints (2.2) defined on the search space are considered strict. There is a low likelihood to find a candidate decision vector that satisfies the constraint for global optimization methods. Thus, in practice, equality constraints are relaxed to a form such as

$$|\mathbf{g}_=(\mathbf{x})| - \boldsymbol{\varepsilon} \leq \mathbf{0},$$

to increase the probability of the constraint to satisfy into regions where $\boldsymbol{\varepsilon}$ is a (small) positive constant vector, and $|\cdot|$ is a component-wise absolute value.

Inequality constraints (2.3) are the most general category such that the previous categories can be expressed using them. Let the following notation represent the violations of constraints

$$G_i(\mathbf{x}) = \begin{cases} \max(0, |\mathbf{g}_{=,i}(\mathbf{x})| - \epsilon_i) & i = 1, 2, \dots, n_=, \\ \max(0, \mathbf{g}_{\leq, i-n_=}(\mathbf{x})) & i = n_= + 1, n_= + 2, \dots, n_= + n_{\leq}. \end{cases} \quad (2.5)$$

Vector $\mathbf{G}(\mathbf{x})$ represents violations of all constraints. The first $n_=$ components of $\mathbf{G}(\mathbf{x})$ represent violations of the specific equality constraints, and n_{\leq} components represent the violations on specific inequality constraints. The global optimization method generates a candidate decision vector that has not been checked if it satisfies the constraints. This vector or the objective function can be adjusted by one of the following methods [34],

- *repair methods* that modify a candidate decision vector
- *problem-specific representation* that re-formulates the objective function in a way such that all possible decision vectors satisfy the constraints

- *penalty methods* transform global optimization objective function into a series of augmented objective functions that converge to the solution of the original objective function evaluation

Repair methods A repair method moves an infeasible decision vector to a boundary inside the feasible search space. Simple methods dealing with boundary constraints can move an element of the decision vector that is out of \mathbb{D} to be inside the \mathbb{D} . In the case when more complex constraints are implemented, the method infers a new value for the element in the decision vector that satisfies a constraint by solving an equation with one unknown. This method can fail because it is problem specific; thus, making it more suitable for enforcing bound constraints.

Problem-specific representation Problem-specific representation methods transform a constraint into the boundary constraint. An example is looking for an optimum within a circle that can be constrained by radius value or have bound constraints using polar coordinates [34]. This method is also problem-specific because the constraints must have a transformation to be represented as bound constraints.

Penalty methods Penalty methods emulate an unconstrained optimization problem with an augmented objective function θ

$$\theta(\mathbf{x}) = f(\mathbf{x}) + r\phi(\mathbf{x})$$

where ϕ is a function that incorporates information about constraints and r is a *penalty factor*. Penalty methods can also be implemented internally into the objective function evaluation as objective function handled constraints. There are two types of categories to incorporate penalty methods:

- *interior decision vector methods* that do not evaluate infeasible decision vectors
- *exterior decision vector methods* that may evaluate infeasible decision vectors

Interior decision vector methods penalize feasible decision vectors that are close to the constraint boundary. This method is implemented by modifying the objective function to incorporate the constraints to make the global optimization unconstrained.

Exterior decision vector methods penalize infeasible decision vectors by using one of the following methods. A death penalty method is a strict form of penalization that rejects any infeasible decision vectors. When a decision vector is rejected by the method, then a new decision vector is generated. The decision vector is checked if it satisfies the constraints and if not, then the process is repeated until a feasible decision vector is generated. The death penalty method is computationally efficient because there is no need for extra evaluation time to calculate the penalty value. However, obtaining a candidate solution can become more difficult because there is no extra information gained from rejecting decision vectors. Thus, the non-death penalty method is commonly used because it uses the constraint violation value.

The other exterior method is the non-death penalty method that applies a penalty cost to infeasible decision vectors. The penalty cost, θ , is based on the constraints (2.5) that the infeasible decision vector did not satisfy. These methods can be implemented in the algorithm or reformulated in the objective function. An example of a method a global optimization algorithm can use for a penalty function is an exact penalty method

$$\theta(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{f} \in \mathbb{F}, \\ +\infty, & \text{otherwise.} \end{cases}$$

A global optimization algorithm uses the exact penalty method to evaluate the objective function and constraints with the decision vector. If it is an infeasible decision vector, then infinity is used as the result of the objective function.

Other forms of the penalty method are embedded in the algorithm of the solver. An example of a constrained global optimization solver that uses a penalty method is Global Search [15]. In Global Search, the method is referred to as the score function that assigns a score based on the objective function evaluation and constraint violations value. This score then is used in the evolution of picking other various decision vectors to evaluate (discussed in Section 2.4.2).

However, in most situations how constraints are handled is tricky to manage because the method or violation cost can either hinder or advance the process of obtaining a minimum. This situation leads to most solvers being unconstrained global optimization solvers and to allow the user to handle the constraints within the objective function. Users can implement

constraints in multiple ways; one simple method is to penalize the objective function evaluation in the objective function and then return it to the solver. Another method is to use a constrained local solver in the objective function to return an objective function evaluation value and an updated decision vector. However, the second method mentioned can be tricky to implement if the solver does not accept updated decision vectors from the objective function. A common method to implement the second method is called the *augmented Lagrangian method* [10] where a global optimization algorithm is used in combination with a constrained local solver. The constrained local solver solves the sub-problem of finding a local feasible decision vector and returns the objective function evaluation and decision vector to the global optimization solver.

Besides the constraints placed on the \mathbb{D} , other algorithms also place constraints internally on the objective function to aid in solving the problem. An example of this is the α -Branch and Bound algorithm developed [3] where sub-domain constraints are placed in the \mathbb{D} to segment sections that enforces the function to be convex. If the objective function section is non-convex, then it is re-segmented until it is convex. An example of this is taking the function shown in Figure 2.1b then applying bound constraints on the function to obtain segmented sections (shown in Figure 2.2) that are treated as sub optimizations within their segmented regions.

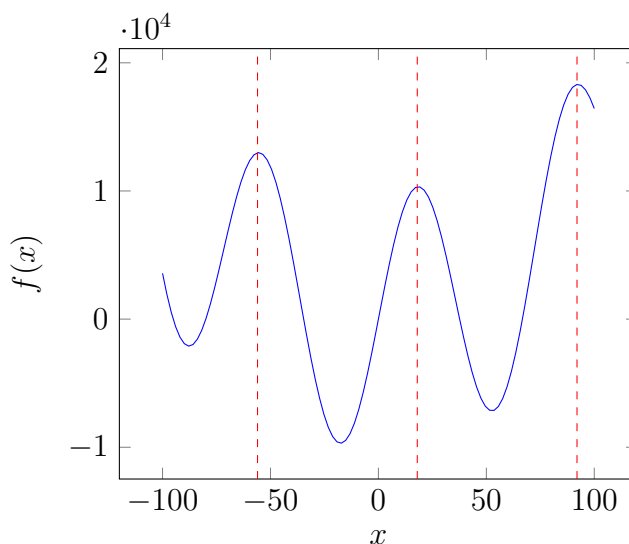


Figure 2.2: Convex segmented regions of a non-convex function.

This enforcement of transforming a non-convex problem to a convex sub-problem is a form of *convex relaxation*, where specific changes to a non-convex function allow it to be solved as a convex function. In this situation, the bounds were changed until the function is convex to allow the algorithm to converge on a local minimum in sub-domains. The algorithm then obtains all the local minimum from the sub-domains to determine the global minimum.

Constraints reformulated into the objective function are a preferred method to implement constraints in a global optimization algorithm because the user can decide how infeasible decision vectors are handled. This reformulation of constraints also allows any global optimization method to be used because the objective function handles the constraints to allow the unconstrained global optimization algorithm to be used.

2.4 Algorithms

In the 1950s, linear optimization problems were solved using linear programming. An example of an algorithm that could solve linear problems is the Simplex algorithm [28] that had a worst-case time complexity of exponential time based on the number of problem variables. Given enough time, linear programming could solve the problem; however, this becomes impractical when the problem is complex or the problem is non-linear. Researchers then began developing other algorithms to solve non-linear global optimization problems by looking at the characterizations of the objective function and the constraints. Global optimization algorithms have traditionally been divided into two main types, deterministic and stochastic.

2.4.1 Deterministic

Deterministic algorithms are guaranteed to find the global minimum by searching the whole domain with tight convergence properties [31]. In 1969, the Branch and Bound algorithm [28] was one of the most well-known algorithms for solving complex problems, like the travelling salesman problem [28] for discrete problems. Other deterministic algorithms include interval optimization [2], algebraic techniques [41], and DIRECT [19] that were used for continuous problems. One potential problem of deterministic algorithms is the computational burden that can be excessive on complex problems.

2.4.2 Stochastic

Stochastic algorithms, developed in the 1970s, use adaptive random methods to obtain a good enough solution for the global minimum in a reasonable amount of time. Unlike deterministic algorithms, stochastic algorithms in most cases cannot guarantee to find a global minimum. Because of this property of stochastic algorithms, further research has been directed towards obtaining better global minima for different classifications of applications [4, 31]. Various sub-categories of stochastic algorithms have appeared, including probabilistic approaches [4], Monte Carlo approaches [4], evolutionary algorithms [4], and metaheuristic methods [8]. Each sub-category targets a different class of applications and shows various improvements on different types of functions. Some well-known algorithms include: Particle Swarm Optimization (evolutionary algorithm) [21], differential evolution (metaheuristic method) [4], genetic algorithm (evolutionary algorithm) [4], cross-entropy method (Monte-Carlo algorithm) [4], and simulated annealing (probabilistic algorithm) [4]. Two algorithms used to solve the applications in Chapter 4 are Global Search and Particle Swarm Optimization.

Global Search

Global Search (GS) is a hybrid heuristic algorithm. The heuristic is the generation of the population decision vectors using the scatter-search algorithm [15]. GS starts by locally optimizing around the initial decision vector, \mathbf{x}_0 , which the user provides to the algorithm. If the local optimization converges, various parameters are recorded, including:

- initial decision vector
- convergent decision vector
- final objective function value
- score value

The *score value* is determined by taking the sum of the objective function value and any constraint violations. If the decision vector is feasible, then the score value is equal to the objective function value. Otherwise, a violation constraint value is summed up for every

constraint not satisfied by the decision vector and multiplied with the objective function value. This score function is a form of a non-death penalty function, the purpose of which is to deter exploration around decision vectors that do not satisfy the constraints.

The algorithm then generates trial decision vectors using the scatter-search algorithm [15]. The scatter-search algorithm is a population-based meta-heuristic algorithm developed to perform a search on the domain. This search on the domain generates new decision vectors of the population based on deterministic combinations of previous decision vectors in the population as opposed to the more extensive use of randomization used in other algorithms. After generating a new set of trial decision vectors, global search then evaluates each decision vector and gives it a scored value. The decision vector with the best-scored values is then optimized by the local solver. The same information is stored in this trial decision vector as the new initial decision vector.

GS then initializes the centre decision vectors and radii of the basins of attraction. The algorithm makes the heuristic assumption that the *basins of attraction* are spherical. Two spheres are centred around the convergent decision vectors of the initial and best trial decision vectors with the radii being the distances from the start decision vectors to the convergent decision vectors of the local optimization. These estimated basins can overlap.

A local solver threshold, l , is initialized to be less than the two convergent objective function values. If the first two decision vectors are infeasible, then the local threshold is set to the score value of the first trial decision vector.

Two counters (one counter per basin) are initialized to zero. These counters are associated with the number of consecutive trial decision vectors that lie within the respective basins of attraction and record the number of times a score value is greater than the local solver threshold.

The algorithm then proceeds to evaluate each trial decision vector using the local optimizer, provided the following conditions hold:

- Condition 1

$$|\mathbf{x}_k[i] - \mathbf{b}[j]| > d \cdot r[j], \quad i = 1, 2, \dots, n \quad j = 1, 2, \dots, m,$$

where $\mathbf{x}_k[i]$ is trial decision vector i at iteration k , $\mathbf{b}[j]$ is basin of attraction centre j ,

n is the total number of trial decision vectors, m is the total number of basins, d is the distance threshold factor (with a default value of 0.75), and $r[j]$ is radius of the basin of attraction j .

- Condition 2

$$\text{score}(\mathbf{x}_k[i]) < l$$

where l is the local solver threshold.

- Condition 3 (optional) $\mathbf{x}_k[i]$ satisfies bound and inequality constraints.

If all conditions are met, then the local solver runs on the trial decision vector, $\mathbf{x}_k[i]$. If the local solver converges, then the global optimum solution is updated, provided one of the following conditions is satisfied:

$$|\mathbf{x}_k^*[m] - \mathbf{x}_k^*[i]| > \tau_x \max(1, |\mathbf{x}_k^*[i]|)$$

or

$$|f_k^*[m] - f_k^*[i]| > \tau_f \max(1, |f_k^*[i]|),$$

where $\mathbf{x}_k^*[m]$ and $\mathbf{x}_k^*[i]$ are convergent decision vectors m and i for the trial decision vectors m and i , respectively; $f_k^*[m]$ and $f_k^*[i]$ are the objective function values for the n and i convergent decision vectors, respectively; τ_x and τ_f are the \mathbf{x} tolerance and function tolerance, respectively, their default values being $1 \cdot 10^{-8}$. If one of the above conditions are satisfied, a global solution object is created or updated to store the best value.

The basin radius and local solver threshold are likewise updated if the local solver converges. The updates are as follows:

- threshold is set to the score value at the trial decision vector
- basin radius is set to the lesser of (i) the distance from $\mathbf{x}_k^*[i]$ to $\mathbf{x}_k[i]$ and (ii) the maximum existing radius (if any)

If the local solver does not run on the trial decision vector due to the conditions not being satisfied, the following Algorithm 1 is executed to update the basin, radius, local solver thresholds, and counters.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a derivative-free algorithm developed in 1995 by Kennedy and Eberhart [21]. This algorithm is inspired by a simplified social swarm model where the algorithm mimics the social behaviour of flocking birds. The social sharing of information is believed to give it an evolutionary advantage [21]. The basic idea of this algorithm is to evolve a number of agents, known as *particles* for PSO-based methods, over a number of iterations. Each particle, i , is evolved towards a randomized combination of its individual best position, $\mathbf{p}[i]$, and the population's global best, \mathbf{p}_g , for every iteration k . This evolved position, $\mathbf{x}_k[i]$, is determined by the particle's velocity

$$\mathbf{v}_{k+1}[i] = \mathbf{v}_k[i] + c_1\varepsilon_1(\mathbf{p}[i] - \mathbf{x}_k[i]) + c_2\varepsilon_2(\mathbf{p}_g - \mathbf{x}_k[i]), \quad (2.6)$$

where c_1 and c_2 are constant factors and ε_1 and ε_2 are random variables that are uniformly distributed in the range $(0, 1)$. The position of the particle is then updated

$$\mathbf{x}_{k+1}[i] = \mathbf{x}_k[i] + \mathbf{v}_k[i]$$

for every iteration of the algorithm.

To avoid premature convergence on local minima or over-exploration of the particles, Shi and Eberhart [33] introduced a new term known as *inertia weight*, w . The inertia weight balances out the premature convergence and over-exploration applications by influencing the previous velocity term

$$\mathbf{v}_{k+1}[i] = w\mathbf{v}_k[i] + c_1\varepsilon_1(\mathbf{p}[i] - \mathbf{x}_k[i]) + c_2\varepsilon_2(\mathbf{p}_g - \mathbf{x}_k[i]). \quad (2.7)$$

This added term showed overall performance increase in the standard PSO algorithm [33]. Later, Clerc [9] used a constriction factor to ensure convergence in the particle swarm. This altered (2.6) to

$$\mathbf{v}_{k+1}[i] = \chi[\mathbf{v}_k[i] + c_1\varepsilon_1(\mathbf{p}[i] - \mathbf{x}_k[i]) + c_2\varepsilon_2(\mathbf{p}_g - \mathbf{x}_k[i])] \quad (2.8)$$

where

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \quad \text{and} \quad \phi = c_1 + c_2, \quad \phi > 4.$$

This equation also prevents particle divergence. The schematic movement of the particle shown in Figure 2.3 follows (2.8).

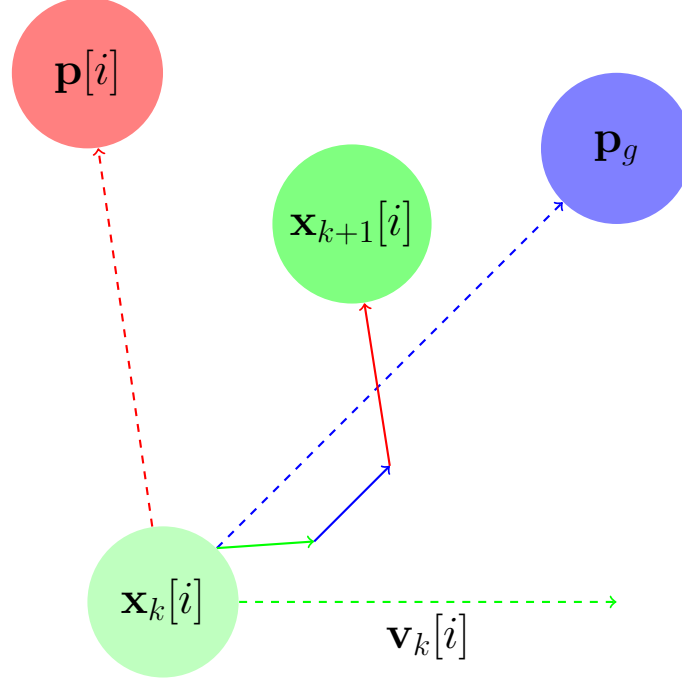


Figure 2.3: Schematic of the particle movement using (2.7).

When compared to the inertia weight (2.6), Shi, and Eberhart found theirs was equivalent in performance [33]. However, current work has shown that (2.6) is a superior method of the standard PSO algorithm [42]. The standard PSO algorithm is shown in Algorithm 2.

Over the past few years, multiple variants of the PSO algorithm have been developed: Collision-free PSO [27], Discrete PSO [22], and Democratic PSO [20]. One variant of PSO, known as Global Convergence PSO, is used to find solutions to the Rational Design of Materials application discussed in Chapter 4.

Global Convergence PSO Van den Bergh and Engelbrecht [40] developed the *Global Convergence PSO* (GCPSO) (also known as Guaranteed Convergence PSO), whereby particles perform a random search around the global best particle within a dynamically adapted search *radius*, ρ . The search radius scales a random factor that is added to the velocity of the particle that most recently updated the swarm's best position [40]. This method reduces

the risk of stagnation and increases local convergence.

GCPSO uses (2.7) and (2.8) to determine the particle's velocity, \mathbf{v}_i , and to update the inertia weight factor, w , over each iteration. The velocity, \mathbf{v}_i , of the particle that most recently updated the best position, \mathbf{p}_g , is updated by

$$\mathbf{v}_{k+1}[i] = -\mathbf{p}_g + \mathbf{x}_k[i] + w\mathbf{v}_k[i] + \rho_k(1 - 2\varepsilon),$$

where ε is a uniformly random number between $(0, 1)$. The initial value of the search radius, ρ_k , is set to one and updated later by

$$\rho_{k+1} = \begin{cases} 2\rho_k, & \sigma_{k+1} > \sigma_c, \\ \frac{1}{2}\rho_k, & \gamma_{k+1} > \gamma_c, \\ \rho_k, & \text{otherwise,} \end{cases}$$

where σ_k and γ_k represent numbers of consecutive successes and failures updated every iteration, k , and σ_c and γ_c represent threshold values. The number of consecutive successes are determined by

$$\sigma_{k+1} = \begin{cases} 0, & \gamma_{k+1} > \gamma_k, \\ \sigma_k + 1, & \text{otherwise,} \end{cases}$$

and the number of consecutive failures are determined by

$$\gamma_{k+1} = \begin{cases} 0, & \sigma_{k+1} > \sigma_k, \\ \gamma_k + 1, & \text{otherwise.} \end{cases}$$

The recommended values for the thresholds for applications with higher dimensions are $\sigma_c = 15$ and $\gamma_c = 5$ [40].

Algorithm 1 Global search algorithm for when the local solver does not run.

```

1:  ▷ When position  $\mathbf{x}_k[i]$  does not satisfy the constraints preventing the local solver from
   running the following code is executed.
2:                                     ▷ Update counters
3: for each basin  $j$  do
4:   if  $\text{InSphereRegion}(\mathbf{b}[j]), \mathbf{x}_k[i]$  then ▷ Checks if the decision vector,  $\mathbf{x}_k[i]$ , is inside
   any of the basins,  $\mathbf{b}[j]$ .
5:      $b_c[j] + = 1$           ▷ If it is add one to the counter associated with basin,  $b_c[j]$ 
6:   else
7:      $b_c[j] \leftarrow 0$           ▷ Otherwise set the counter to zero.
8:   end if
9: end for
10: if  $\text{score}(\mathbf{x}_k[i]) \geq l$  then ▷ If the score value of the position,  $\mathbf{x}_k[i]$  is greater than the
   local solver threshold,  $l$ 
11:    $l_c[i] + = 1$     ▷ then increment the threshold counter  $s_c$  for the position  $\mathbf{x}_k[i]$  by one.
12: else
13:    $l_c[i] \leftarrow 0$           ▷ Otherwise, set the score counter to zero.
14: end if
15:                                     ▷ React to large counter values
16:                                     ▷ MaxWaitCycle and BasinRadiusFactor are settings for Global Search
17: for each basin  $j$  do
18:   if  $b_c[j] == \text{MaxWaitCycle}$  then    ▷ If the basin counter is equal to MaxWaitCycle
19:      $r[j] \leftarrow r[j] \times (1 - \text{BasinRadiusFactor})$   ▷ then multiply the basin radius,  $r$ , by
      $1 - \text{BasinRadiusFactor}$ 
20:      $b_c[j] \leftarrow 0$           ▷ and reset basin counter to zero.
21:   end if
22:   if  $l_c[j] == \text{MaxWaitCycle}$  then    ▷ If the threshold counter equals MaxWaitCycle
23:      $l \leftarrow l + P_f \times (1 + |l|)$  ▷ increase the threshold by adding penalty threshold factor,
      $P_f$ , multiplied by  $1 + |l|$  to the threshold
24:      $l_c[j] \leftarrow 0$           ▷ reset the threshold to zero.
25:   end if
26: end for

```

Algorithm 2 Particle Swarm Optimization

```
1: for each particle  $i$  do
2:   initialization  $\mathbf{x}_k[i]$ ,  $\mathbf{v}_k[i]$ ,  $\mathbf{p}[i]$             $\triangleright$  random value for  $\mathbf{x}_k[i]$  and  $\mathbf{v}_k[i]$   $\mathbf{p}[i] \leftarrow \mathbf{p}[i]$ 
3:   Evaluate  $f(\mathbf{x}_k[i])$                                 $\triangleright$  evaluate the objective function at  $\mathbf{x}_k[i]$ 
4:   Update  $\mathbf{p}[i]$                                         $\triangleright$  update if  $f(\mathbf{x}_k[i]) < f(\mathbf{p}[i])$ 
5: end for
6: while not termination condition do
7:   for each particle  $i$  do
8:     update  $\mathbf{p}_g$                                         $\triangleright$  update if  $f(\mathbf{p}_g) < f(\mathbf{p}[i])$ 
9:     calculate  $\mathbf{v}_k[i]$                                 $\triangleright$  Using one of the PSO velocity equations
10:     $\mathbf{x}_k[i] = \mathbf{x}_k[i] + \mathbf{v}_k[i]$ 
11:    Evaluate  $f(\mathbf{x}_k[i])$ 
12:    update  $\mathbf{p}[i]$ 
13:   end for
14: end while
```

CHAPTER 3

SOFTWARE

In this chapter, two pieces of software are presented as support software for global optimization applications. Support software can add flexibility, features, and performance increases to the global optimization solver or objective function. Some examples of support software for global optimizations are the CUDA library for parallel programming on Graphics Processing Unit (GPU) [18] and MapReduce to distribute a hybrid PSO-GA algorithm [32]. Another support software package that utilizes *public-resource computing and storage* is the Berkeley Open Infrastructure for Network Computing (BOINC) [5]. Public-resource computing uses a distributed system of public computers to compute scientific calculations. BOINC distributes tasks to client computers that then report back to the BOINC server to store the results. An extra step that BOINC takes is validating results by repeating the task on another client machine. This step is done in anticipation of corrupted files being sent over the network or failure in the task. This redundancy can be an inefficiency to global optimization algorithms by evaluating the objective function twice and providing no extra information to the algorithm or performance increase.

The first piece of support software discussed in this thesis is Computefarm. Computefarm is a distributed local machine resource system that distributes objective functions to local client machines that then report back to a server machine. It has a python interface using C as the back end to run in parallel using the POSIX threads library. The software is built with fault tolerance in the event a client machine fails. One step Computefarm omits, but is taken in BOINC, is the validation step to keep it lightweight for global optimization use. It is used with GCPSO in pythOPT to solve the Rational Design of Materials application discussed in Chapter 4.2.

The other piece of support software developed for this thesis is the Optimization Database.

It is a relational database that stores intermediate data of the global optimization process. This software is integrated with a notification emailing script and a website to show periodic and real-time updates on the global optimization progress. This database is used to monitor the two applications discussed in Chapter 4.

3.1 Computefarm

3.1.1 Motivation

The evaluation time of the objective function can hinder the global optimization time. This delay is commonly handled by parallelizing the global optimization algorithm. In some cases, researchers develop various parallel methods to speed up the evaluation time of the objective function [36]. If the objective function evaluation requires a lot of computational resources, a supercomputer is typically used [13]. However, supercomputer resources are not readily available or are costly to obtain. Therefore, methods that use distributed systems like MapReduce [32] or BOINC [13] become an option to use grid computation or public resources. To optimize the use of computational resources to evaluate multiple objective functions simultaneously, Computefarm distributes objective function evaluations to client machines using *local-resource* computing.

Distributing the objective function evaluations out to client computers reduces the number of computer resources consumed and evaluates multiple decision vectors simultaneously. Computefarm also handles failures in the client machines. If a client disconnects or fails to complete the objective function evaluation, Computefarm reassigns the evaluation to another client. This contrasts with the classical case, when a single machine faces a failure on its own, the entire optimization process is interrupted and needs to be restarted. Computefarm provides parallelization, reduced resource contention, and fault tolerance in failures in objective function evaluations for the global optimization.

3.1.2 Requirements

ComputeFarm is a distributed system that delegates tasks to multiple client computers. In the case of global optimization, a server distributes objective function evaluations to client computers. Three things that are required for using ComputeFarm:

- the port number for socket connection (user-provided)
- the script name of the objective function (user-provided)
- the list of population decision vectors at which to evaluate the objective function on client machines (algorithm-provided)

The first two items are passed into the solver and subsequently passed along to ComputeFarm for the initialization phase. The last item is passed in by the algorithm according to the metric it uses to determine which decision vectors are to be evaluated at each iteration step.

3.1.3 Structure

The implementation of ComputeFarm is based on the client-server model; the client machines request objective function evaluations from the server that then delegates the decision vectors out to the client machines. This model parallelizes the evaluation step of the global optimization algorithm by distributing out multiple objective function evaluations simultaneously. Once the client machines have evaluated all decision vectors, the results are returned to the solver to be evolved and repeated until the termination condition is satisfied.

In the situation of losing connection with the local client machine, the server receives return value of zero from the TCP socket indicating a client machine has disconnected. The decision vector assigned to the client machine that has disconnected is returned to the queue. This decision vector is later delegated to one of the client machines waiting to evaluate another decision vector. The global optimization solver does not need to restart if a client computer disconnects; this handles the case if a disconnect occurs on the client side in the ComputeFarm software. To guarantee the maximum number of clients are available for the optimization process, ComputeFarm reawakens client machines periodically. Figure 3.1 shows the flow of a global optimization algorithm using ComputeFarm.

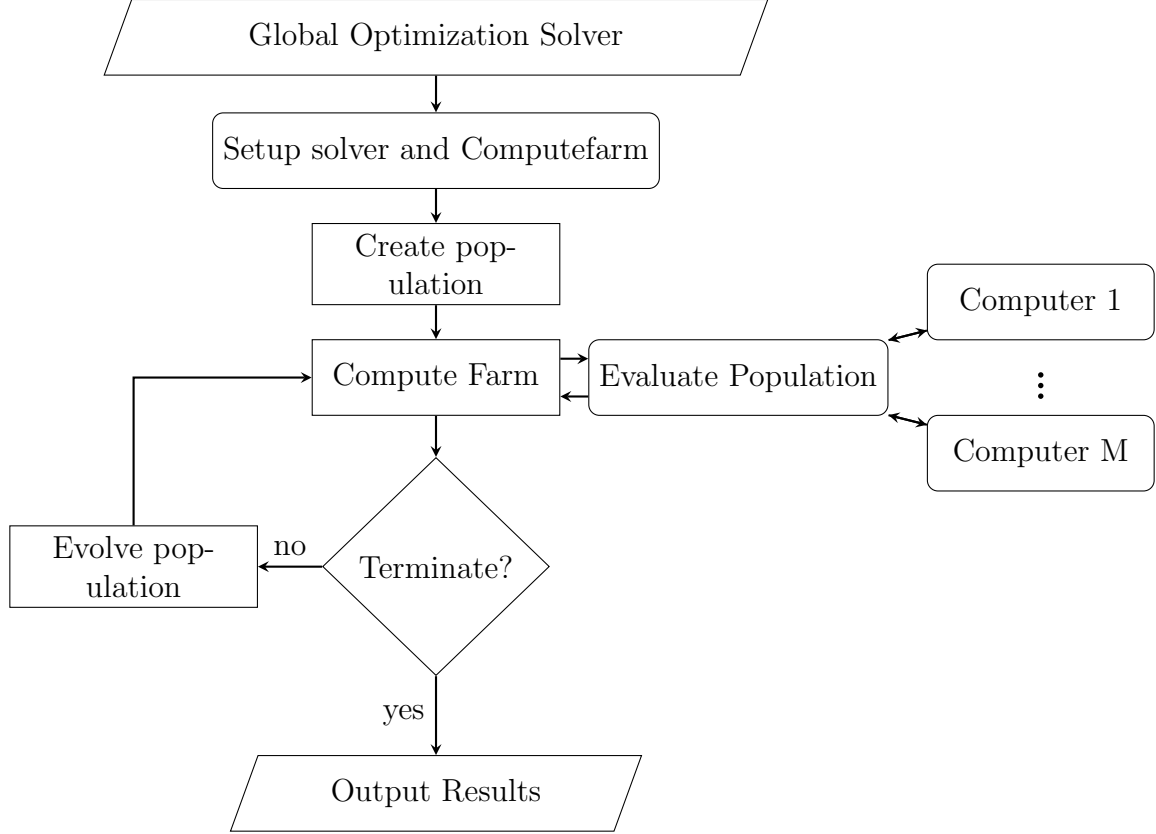


Figure 3.1: Process of a global optimization algorithm using Computefarm.

ComputeFarm takes a list of population decision vectors and distributes them to various client computers, which return a list of results to the global optimization algorithm to further proceed in the process. In this thesis, the PSO algorithm that leverages ComputeFarm to solve the application is described in Chapter 4. The original PSO algorithm is described in Algorithm 2, and the ComputeFarm version of PSO is described in Algorithm 3 that substitutes the evaluation step of the PSO algorithm with ComputeFarm.

Algorithm 3 is implemented in the software package `pythOPT`, as a variant to the PSO algorithms known as ComputeFarm Particle Swarm Optimization (CFPSO). The interface and usage of CFPSO is shown in Appendix A.

The software step of ComputeFarm, Figure 3.2, shows the overall process in the software package of ComputeFarm that is being called to evaluate a population of decision vectors.

During the setup phase of the ComputeFarm software, three *POSIX threads* are used to run the following functions:

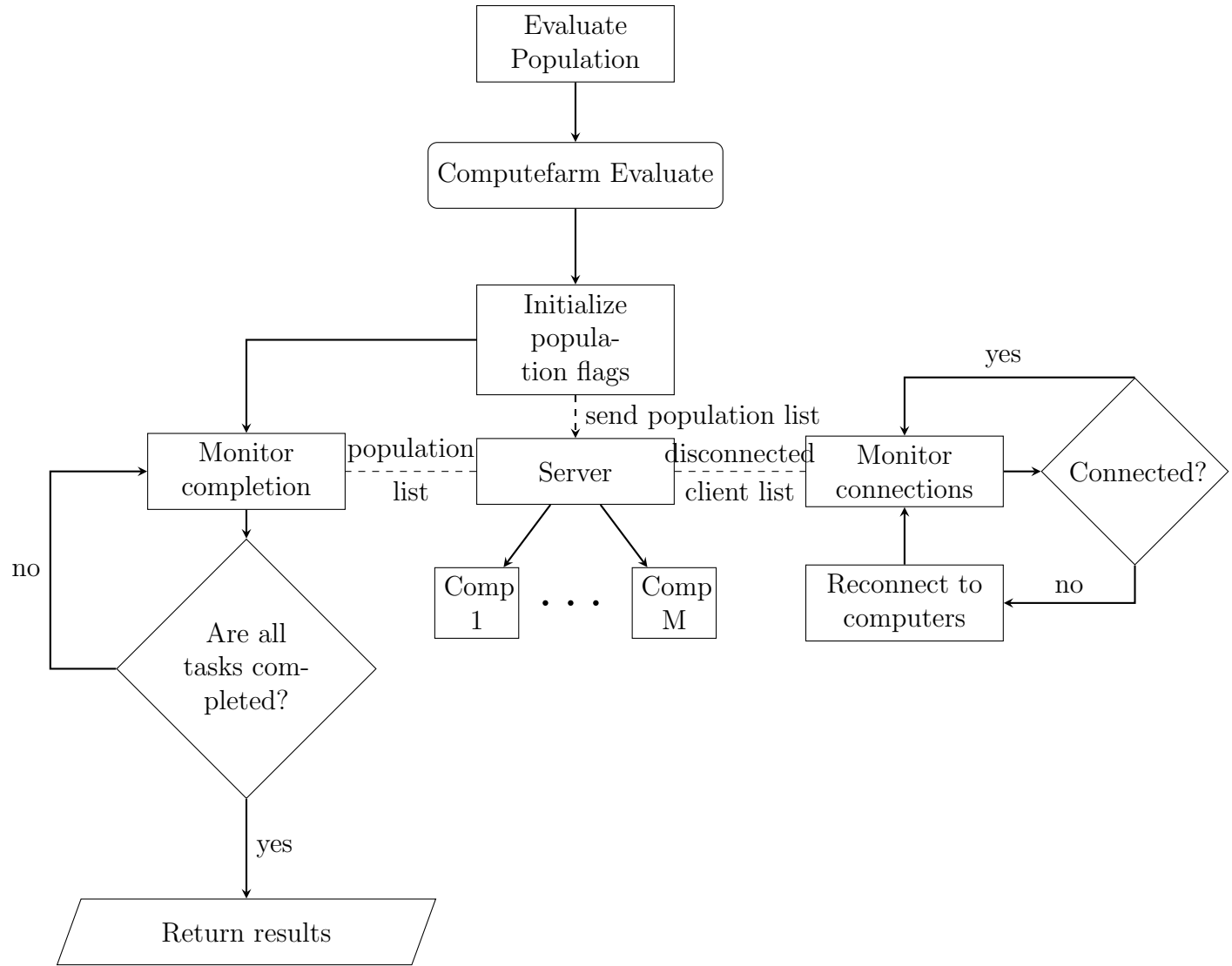


Figure 3.2: Process of a global optimization algorithm using ComputeFarm to evaluate positions.

- Monitor Completion
- Server
- Monitor Connections

Threads are used in this software because of their shared memory properties, which allow for a form of message passing in the program. Each thread function is attached to a single process that monitors a list to determine further actions. The lists that are shared in memory between the three threads are the medium for message passing in the program.

The Monitor Completion function monitors whether all tasks are completed before returning the results to the solver. The Server function creates a TCP socket that binds to any client computers communicating on the same port. TCP sockets were chosen because of their reliable connection-handling capabilities. Threads are generated for each connected client; this allows for concurrency in the server system. Each connection is taken care of by a connection handler that assigns itself to a client machine by receiving the hostname of that machine. Once the hostname is obtained, the connection handler changes a client-assigned flag for that machine from zero to one; it then continues to monitor the decision vector list that contains information on the decision vector called a particle in the code. The particle structure includes:

- particle number
- particle decision vector
- function value
- length of a decision vector (requirement of C arrays)
- assigned flag
- completion flag

Once a connection handler finds an assigned decision vector, an assigned flag that equals zero is changed to one. *Mutexes* are used to ensure mutual exclusion for the assignment of particles; this prevents multiple connection handlers from re-evaluating the same particle. Once a connection handler has assigned itself a particle, it then transmits the particle position information to the client computer using the TCP socket. This portion of the program is written in the programming language C; the length of a decision vector is sent to the client because it is a requirement for using C arrays. The connection handler then waits to receive a message from the client. This message contains the resulting objective function evaluation result. The connection handler then stores the result in the particle structure. It then changes the completion flag from zero to one indicating the particle has been evaluated successfully. This flag prevents other connection handlers from re-evaluating the same particle. The

monitor completion thread counts the number of completion flags set to one to determine when the population is successfully evaluated. The results are then sent back to the global optimization solver.

The connection handlers are left in an idle state waiting on a new population of decision vectors while the global optimization solver is creating a new population. Once a new population is passed to Computefarm, the particle structures are reinitialized obtaining new decision vectors. The above process is repeated until a termination condition is satisfied. This condition is set in the global optimization solver that then calls Computefarm's terminate function that gracefully closes all connection handlers, frees up any used memory, and exits the threads.

In the situation of a failure in the client machine, the connection handler reverses the particle's assigned flag and the client-assigned flag back to zero and exits. This change in the flag values adds the particle structure back into the queue and the client machine name into the list of machines that need to be reawakened. The workflow of the connection handler is shown in Figure 3.3 and the Monitor Connection loop is shown in Figure 3.4.

The Monitor Connection loop shows the process of reawakening disconnected clients. This loop is embedded into a separate thread function, *Reawaken Clients*, that attempts to awaken the disconnected clients every T minutes (the default set to ten minutes). This reawakening step provides the maximum number of client machines are connected to the server every T minutes from startup. Disconnected clients are determined by the assigned flag in the client structure containing the hostname of the client. When the assigned flag is zero, the Reawaken Clients function attempts to awaken that client machine.

Algorithm 3 Computefarm Particle Swarm Optimization.

initialize Computefarm ▷ initializes the setup of Computefarm

2: **for** each particle i **do**

initialization $\mathbf{x}^*[i], \mathbf{v}[i], \mathbf{p}[i]$ ▷ generate random values for particle positions, $\mathbf{x}[i]$,
and velocity, $\mathbf{v}[i]$

4: $\mathbf{p}[i] \leftarrow \mathbf{x}[i]$ ▷ Set the local best particle position, $\mathbf{p}[i]$

end for

6: **Evaluate** Computefarm(\mathbf{X}) ▷ evaluate the initial population of particle with
Computefarm

8: **while** not termination condition **do**

for each particle i **do**

10: **if** $f(\mathbf{x}[i]) > f(\mathbf{p}[i])$ **then**

Update $\mathbf{p}[i]$

12: **end if**

if $f(\mathbf{p}_g) < f(\mathbf{p}[i])$ **then** ▷ update global best position, \mathbf{p}_g

14: **update** \mathbf{p}_g

end if

16: **calculate** $\mathbf{v}[i]$ ▷ using PSO velocity equation (2.6)

$\mathbf{x}[i] = \mathbf{x}[i] + \mathbf{v}[i]$

18: **end for**

Evaluate Computefarm(\mathbf{X})

20: **end while**

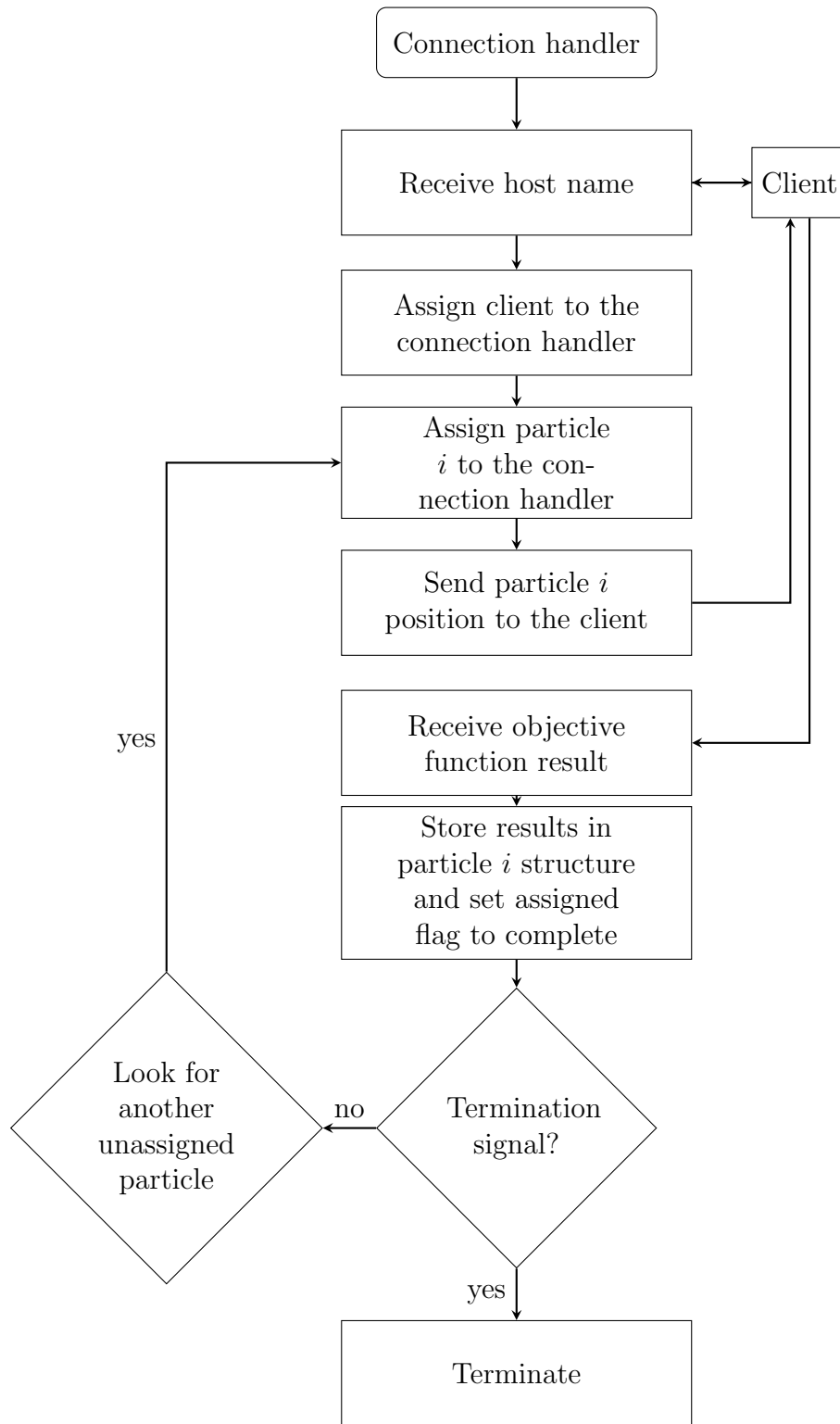


Figure 3.3: Computefarm connection handler work flow.

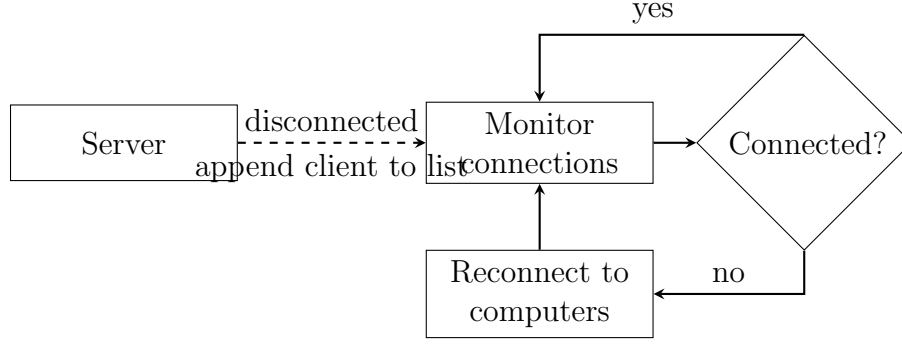


Figure 3.4: The workflow of monitoring lost client connections.

After the setup of ComputeFarm, the Server and Monitor Connections threads are kept alive while the Monitor Completion function is called by the solver. The Monitor Completion function re-initializes the particle list with new positions and resets all other information on the particle. Meanwhile, the process of the Server and Monitor Connections continuously waits to evaluate particles or reconnects to client machines. Once every particle is evaluated, the Monitor Completion returns the results to the solver. This step is repeated multiple times before the final phase, where the solver completes its optimization process and terminates. The termination step occurs when the solver calls the termination function that frees up memory and gracefully closes the ComputeFarm threads. This termination call prevents memory leakage and zombie threads.

3.1.4 Performance analysis

In a simple experiment using SPSO with an objective function evaluation time of 32 minutes (optimization and computer specification shown in Appendix A), ComputeFarm shows a reduction of about fifteen times total optimization time compared to SPSO; see Table 3.1.

ComputeFarm’s distribution of objective function evaluations to multiple local client computers incurs about 3.17 minutes of overhead for each call. This distributive scheme enables a parallel performance increase to the global optimization algorithm during the evaluation phase of the objective function.

Method	Average time (minutes) for 1 objective function evaluation	Average time (minutes) for 1 iteration (30 objective function evaluations)
SPSO	32.65	979.28
SPSO with ComputeFarm	32.65	35.82

Table 3.1: ComputeFarm performance comparison to SPSO in pythOPT.

3.2 Optimization Database

3.2.1 Motivation

The Optimization Database is used to deal with various challenges that emerge in the global optimization process. One set of challenges has to do with monitoring the optimization to determine whether the problem is solved. A standard method of monitoring the global optimization process is to have each iteration printed or saved in a log file. Files can be challenging to read and parse quickly. They also require a strict organizational system to ensure storage is kept central and available to users. A database can be used to avoid these challenges of a file system. Databases place a strict policy on the organization of how information is stored to allow for quick query calls; for example, to obtain the minimum value during a global optimization process a simple query “SELECT MIN(f) FROM problem_table”. This query allows for a quick search to be done in the database to obtain the minimum objective function evaluation value for a given problem stored in the problem table. A database is typically stored on servers allowing for central use for multiple to limited user access ensuring easy and secure access to the information. In this section, a specific database schema is developed with a software package called Optimization Database. The database is a relational database written in PostgreSQL, and the interface to the database is written in Python (See Appendix B.0.1 for the application interface). The Optimization Database has three saving

policies implemented from which the user can choose (see setup script in Appendix B):

- best value
- every evaluation
- every n evaluations

Best value The best value policy only stores the minimum (or maximum specified by the user; see Appendix B) objective function evaluation in comparison to the previously stored value. This policy is done by a quick query to obtain the minimum objective function value for that instance that is then compared to the currently being saved objective function evaluation. If it is less than the previous objective function evaluation, then it is inserted into the database, and the check-in is updated. Otherwise, the check-in time updated for status checking purposes (discussed later in this section) and the objective function evaluation is not stored in the database. This policy is the default policy for the Optimization Database because it does not store every evaluation to save on space in the database.

Every evaluation The every evaluation policy inserts every objective function evaluation into the database. This policy is used in the case of monitoring the global optimization solvers evolution over the number of iterations.

Every n evaluations Every n evaluations policy is the same as the every evaluation policy except that it stores the objective function evaluation every n evaluations. This policy is used when storing every evaluation is too much for the database server, but the user still wants to monitor some information on global optimization evolution.

Another advantage of using a database is that the transaction of inserting data is only completed if all data can be stored in the database. This advantage mitigates corruption in the data when they are being stored. If the transaction fails, an error is reported, and C attempts are made to reconnect (default is two attempts) and insert the data into the Optimization Database. If the attempts are unsuccessful to reconnect to the database, the database interface prints out a warning message informing the user that it could not connect to the database. The warning message allows the user to be aware of potentially unsaved

data and that something has gone wrong with the database. The optimization process is not disrupted by the warning message. When the database is called again to save data, the interface attempts again to reconnect to the database. This reconnection step provides fault tolerance in the data storage without affecting the overall process of the global optimization. Examples that monitoring the global optimization process is useful includes:

- premature convergence
- performance testing
- obtaining continuous results

Premature convergence Global optimization algorithms are susceptible to premature convergence or stagnation when the process gets trapped in a local minimum. The Optimization Database can monitor the progress of the optimization by storing the time and evaluation number of each saved value. Using the email notification script or accessing the database can indicate if progress has stagnated. The user can further decide how to resolve this problem, for example, starting other instances of the global optimization algorithm to see if any other progress can be made. Another feature of the database is that it stores the activity of global optimization instances. If the optimization stops, then the status is set to “inactive”, and the user is notified by the email script or can check on it in the database.

Performance testing It is common for global optimization algorithms to be bench-marked on solving various classes of problems for performance analysis. The Optimization Database stores the evaluation number, extra information on the objective function or global optimization, and the activity of the process. This storage scheme can be used to analyze the convergence of global optimization algorithm and any other extra information the user wishes to store in the database.

Obtaining continuous results Some global optimization algorithms only report their results at the end of the optimization process; this can prevent the user from becoming aware if a solution is obtained earlier in the optimization. The Optimization Database is used to monitor and notifies the user if a desired objective function value is obtained.

The Optimization Database notifies the user by email when the desired objective function evaluation value is obtained and stored in the database.

Another challenge of global optimization is obtaining the K best solutions. Global optimization is implemented to solve for a global minimum; it is less frequent for solvers to return the K best global solutions. The Optimization Database is therefore used to sort through the data to obtain the K best solutions. This feature is shown to be useful in Chapter 4.2 for the Rational Design of Materials project. The metastable structures of crystals are of particular interest because of their properties; diamond, for example, is a metastable structure of carbon, with the stable structure being graphite. In a global optimization, graphite is the global minimum because it has the lowest total energy. Diamond, with the second lowest total energy, has the property of being very hard and is used in multiple research experiments to define hardness with respect to other structures. The Optimization Database is used to store the K lowest energy crystal structures for various compounds.

Another data-storing problem for global optimization is obtaining extra information on the objective function. Extra information may include:

- sorting information on various instances (e.g., the time duration to correct for errors in a quantum error correction gate, as discussed in Chapter 4.1) or
- information on the data (e.g., the symmetry group of a predicted crystal structure as discussed in Chapter 4.2).

3.2.2 Requirements

The database is primarily operated by a single user to keep track of the progress of a solver on a problem of interest. Primary information that is needed:

- PostgreSQL database information
- problem name
- global optimization settings

3.2.3 Database schema

The Optimization Database is implemented for users with no prior knowledge of databases. If the user does not already have a local or remote database set up, the software creates a local PostgreSQL database. It then automatically produces two tables: the settings table, and the problem table (if they do not already exist). The former, stores information about the settings used for the global optimization, with the following default columns:

- global optimization method name
- lower bound on the search space
- upper bound on the search space
- seed value for global optimization method (if a stochastic method is used)
- a note describing the optimization problem and process

Columns that are included and maintained by the database are:

- primary key ID
- status
- check-in time

The primary key is used to associate an optimization instance data to the problem's settings stored in the problem table. The status column is updated by the database to monitor whether a simulation is active or inactive. This column is used in combination with the email notification package to notify the user when a problem is inactive. The status of an instance is determined by the check-in time. When the database is updated or inserted into, the check-in time is updated to the current time. If the instance has not checked-in with the database for T hours (default is set to 24 hours), then the Optimization Database sets the status to inactive. These features of the Optimization Database, in combination with an automatic email component, allow the user to be regularly notified regarding the status and results of the optimization.

The problem table is used to record the data on the objective function. By default, it stores:

- the ID of the instance (foreign key)
- \mathbf{x} position
- f (objective function evaluation result)
- evaluation number
- insertion time

The user can also opt for additional columns to store other data about the objective function evaluation. This extra information can be utilized for faster sorting methods between instances or properties of the simulation. Figure 3.5 represents the database schema used by the software. The settings table is associated with multiple instances of the problem tables by the foreign key. The foreign key is a type of ID stored in a database that references the primary key in another table. This type insures that the instance has to exist in the table with the primary key before inserting into another table that is using the same ID as the foreign key. It also ensure consistency between tables to allow querying for multiple pieces of information that is stored in multiple tables.

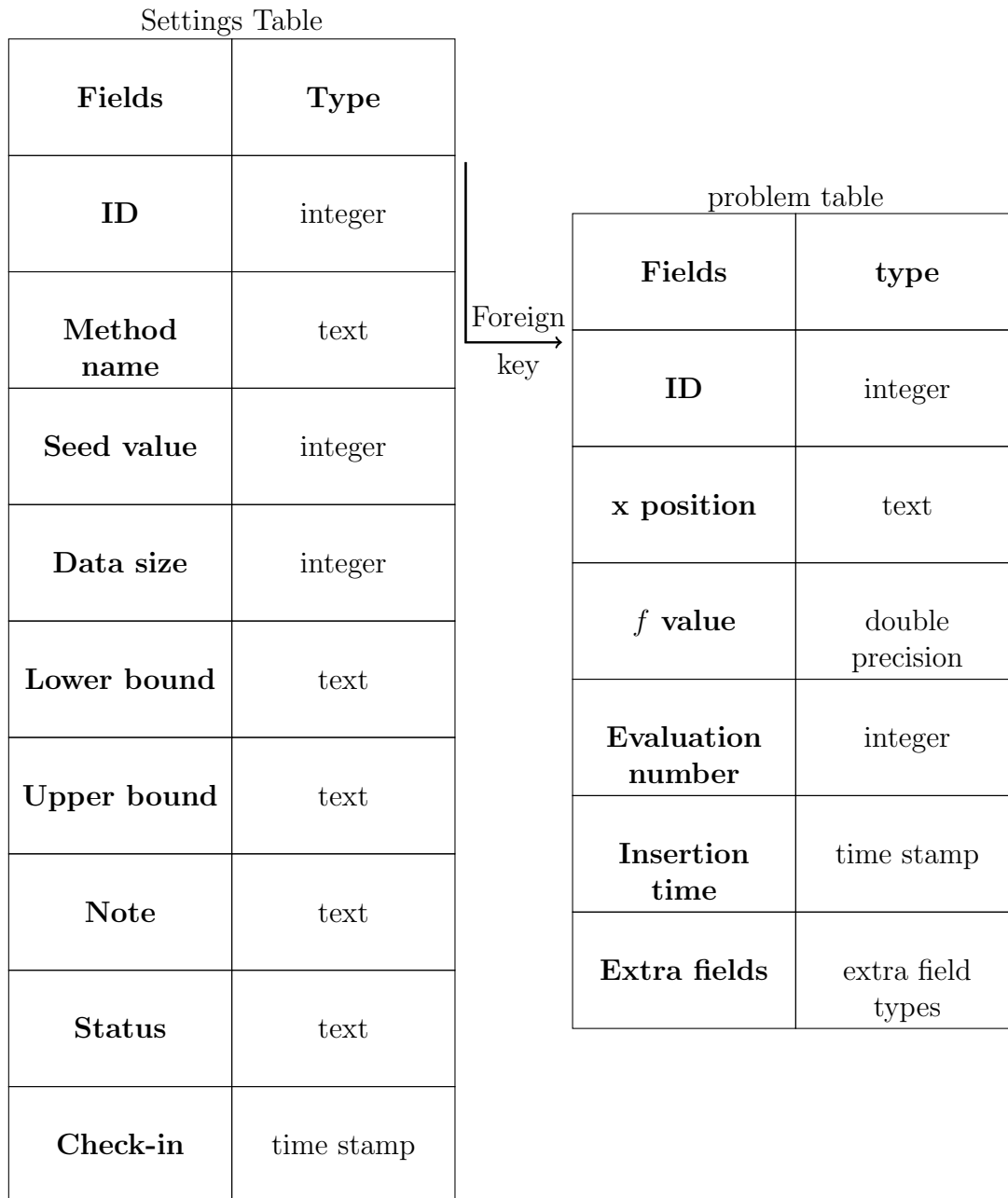


Figure 3.5: Optimization Database schema. One settings table to many problem tables.

3.2.4 Monitor schemes

The Optimization Database software package has two monitoring schemes. The first monitoring scheme is an emailing notification system that is written in python. This feature sends out a prototyping email to the users daily, at a given hour, to notify them of the progress of the global optimization. The email includes the following information:

- start time, when the instance of the optimization was started
- end time, when the last best objective function evaluation value is updated
- previous best objective function evaluation value
- current best objective function evaluation value
- the global optimization name
- extra information on the instance and simulation
- status of the global optimization with the given primary id

Figure 3.6 shows how the information that is displayed in a table format for various instances of a simulation for one global optimization problem.

The start time indicates when the instance is started, and the end time is given when the last updated best value is obtained. The previous best objection function evaluation is stored at the beginning. The email notification queries the Optimization Database to obtain the current best objection function evaluation. If the current best objection function evaluation is better, as determined by the user by indicating a maximum or minimum flag (shown in the setup script described in Appendix B), then it is stored as the best objective function evaluation in the email, and the previous value is stored as the previous best objective function evaluation. Extra fields can be added to the email notification script to monitor other information about the problem. In Figure 3.6, the time duration, T , value is also present in the email to order the instances by the time duration for the Quantum Error Correction problem (described in Section 4.1). The extra fields can be used for ordering the instances (default is id) in the periodic email, for example Figure 3.6 orders the 4-qubit

4 Qubit ☐ Inbox x

NumericalSimulationsLab@cs.usask.ca
to mts299, spiten, wael192

10:06 AM (45 minutes ago)

Start time	End time	Prev f	Best f	Method	T	Note	Status (id)
August 31, 2017 13:27	September 24, 2017 00:11	0.99966925	0.99966925	Matlab globalsearch	62	T=62, 4 qubit. Computemoler2	inactive (2287)
January 31, 2018 22:04	January 31, 2018 22:04	0.99980870	0.99980870	Matlab globalsearch	63	T=63 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=63 . Computer:	inactive (2940)
March 14, 2018 15:54	March 14, 2018 15:54	0.99980870	0.99980870	Matlab globalsearch	63	T=63 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=63 . Computer: moler2	active (3000)
August 31, 2017 13:27	August 31, 2017 13:27	0.99970883	0.99970883	Matlab globalsearch	64	T=64, 4 qubit. Computer: moler2	inactive (2285)
September 05, 2017 19:34	September 05, 2017 19:34	0.99941701	0.99941701	Matlab globalsearch	65	T=65, 4 qubit. Computer: Moler1	inactive (2338)
February 01, 2018 09:33	February 01, 2018 09:33	0.99915824	0.99915824	Matlab globalsearch	69	T=69 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=69 . Computer: moler1	inactive (2963)
March 14, 2018 15:55	March 14, 2018 15:55	0.99915824	0.99915824	Matlab globalsearch	69	T=69 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=69 . Computer: moler2	active (3001)
January 31, 2018 22:03	March 14, 2018 09:59	0.99979386	0.99979386	Matlab globalsearch	70	T=70 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=70 . Computer: moler2	inactive (2938)
January 31, 2018 22:11	January 31, 2018 22:21	0.99977724	0.99977724	Matlab globalsearch	70	T=70 4 Qubit Piecewise pulse using Frobenius without squaring with a start value at the best of T=70 . Computer: moler2	inactive (2943)
January 31, 2018 22:23	January 31, 2018 22:25	0.99977724	0.99977724	Matlab globalsearch	70	T=70 4 Qubit Piecewise pulse using arccos(fidelity) with a start value at id=2847 . Computer: moler1	inactive (2950)
February 01, 2018 08:29	February 01, 2018 08:41	0.99977724	0.99977724	Matlab globalsearch	70	T=70 4 Qubit Piecewise pulse using arccos with a start value at the best of T=70 . Computer: moler2	inactive (2956)
March 14, 2018 12:04	March 14, 2018 12:12	0.99977724	0.99977724	Matlab globalsearch	70	T=70 4 Qubit Piecewise pulse using arccos(fidelity) with a start value at id=2950 . Computer: moler1	active (2990)
March 14, 2018 15:54	April 04, 2018 10:02	0.99362116	0.99362228	Matlab globalsearch	70	T=70 4 Qubit Piecewise pulse using arccos with a start value at the best of T=70 . Computer: moler2	active (2999)
March 14, 2018 15:55	April 04, 2018 10:05	0.99566802	0.99566906	Matlab globalsearch	70	T=70 4 Qubit Piecewise pulse using Frobenius without squaring with a start value at the best of T=70 . Computer: moler2	active (3002)

Figure 3.6: Snapshot of a periodic email notification displaying the information stored in the Optimization Database for the 4-qubit problem.


instances by the column name T. An email template script sets up all the settings in the periodic email notifications (see Appendix B for example of email template). The Note is the description the user stores in the Optimization Database about the problem and instance that they are trying to solve. This description allows for better clarity on differences between instances, which machine they might be optimized on, and what other methods may be used in the objective function. The description can be anything the user wishes to describe about the instance.

The status indicates the current running status of the global optimization and the primary ID associated with that instance in Optimization Database. There are three possible instances:

- active, the instance is still currently running
- inactive, the instance is not currently running
- finished, the instance has completed the global optimization process

For better readability, the status is colour coded. Active is light green if the instance has started recently (default three hour window), and then turns dark green later on. The

inactive status is red to indicate to the user something has occurred that may need their attention. Finished is black to indicate the global optimization is completed. Inactivity is determined by the package by checking the last check-in time stamp and comparing it to the current time. If the time difference is greater than a set inactive time value (default three hours), then package stores the status as inactive in the Optimization Database and in the package. When an instance is detected as inactive, the email package notifies the users right away that an instance has become inactive; Figure 3.7 shows the inactive email the users get when an instance is inactive.



Start time	End time	Prev f	Best f	Method	T	Note	Status (id)
January 31, 2018 22:07	March 14, 2018 00:47	0.97097298	0.97105713	Matlab globalsearch	70	T=70 4 Qubit Error function pulse using fidelity with a start value at the best of T=70 . Computer: moler2	inactive (2941)
January 31, 2018 22:57	February 18, 2018 09:45	0.96998989	0.97003622	Matlab globalsearch	70	T=70 4 Qubit Error function pulse using Frobenius without squaring with a start value at the best of T=70 . Computer: moler1	inactive (2952)
February 05, 2018 09:19	February 20, 2018 14:55	0.96895315	0.96895615	Matlab globalsearch	70	T=70 4 Qubit Error function pulse using arcc with a start value at id=2934 . Computer: moler1	inactive (2977)

Figure 3.7: Inactive email notification.

When an instance has finished, the status is changed in the Optimization Database, and the package detects the change by querying the Optimization Database (every hour). A completion email is sent out to the users notifying them that the instance has completed; Figure 3.8 shows the completion email the users receive when the global optimization is completed. A completion email is also sent out when the user specifies in the setup script (shown in Appendix B) a value of interest. If users know a specific objection function evaluation value they wish to obtain, then the email package notifies the users when an instance has obtained this value. An example of this is in the Quantum Error Correction problem, where for any given instance a objection function evaluation greater than 99.99% is required value to solve the problem. The email package then notifies the users when an instance has achieved an objection function evaluation greater than 99.99%; this is further discussed in Section 4.1.

The other monitoring scheme is a prototype real-time monitor that displays the same

Optimization is complete!

Start time	End time	Prev f	Best f	Method	T	Note	Status (id)
November 28, 2017 11:45	November 28, 2017 14:29	0.99992643	0.99992643	Matlab globalsearch	25	T=25 3 Qubit Error function pulse using fidelity with a start value at the best of 25 starting at pchip of T=26 solution. Computer: moler1	finished (2806)

Figure 3.8: Completion email notification.

information as the daily email notification, however, on a webpage written in PHP. This webpage (shown in Figure 3.9) allows for constant checks on the global optimization progress if the user wants to monitor the instance more closely than by daily email notifications. The mechanisms for the status changes and information gathering is the same as the email notification except that is done constant for real-time displaying of the information.

Numerical Simulation Optimization Database

3 Qubit	3 Qubit erf	4 Qubit	4 Qubit erf	5 Qubit		
4-Qubit						
Start time	End time	Best f	Method	t	Note	Status
Aug 31,2017 13:27	Sep 24,2017 00:11	0.999669253826141	"Matlab globalsearch"	62	"T=62, 4 qubit. Computermoler2"	inactive (2287)
Jan 31,2018 22:04	Jan 31,2018 22:04	0.999808699244	"Matlab globalsearch"	63	"T=63 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=63 . Computer: "	inactive (2940)
Mar 14,2018 15:54	Mar 14,2018 15:54	0.999808699244	"Matlab globalsearch"	63	"T=63 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=63 . Computer: moler2 "	active (3000)
Aug 31,2017 13:27	Aug 31,2017 13:27	0.999708831310272	"Matlab globalsearch"	64	"T=64, 4 qubit. Computer: moler2"	inactive (2285)
Sep 05,2017 19:34	Sep 05,2017 19:34	0.999417006969452	"Matlab globalsearch"	65	"T=65, 4 qubit. Computer: Moler1"	inactive (2338)
Feb 01,2018 09:33	Feb 01,2018 09:33	0.999158236756	"Matlab globalsearch"	69	"T=69 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=69 . Computer: moler1 "	inactive (2963)
Mar 14,2018 15:55	Mar 14,2018 15:55	0.999158236756	"Matlab globalsearch"	69	"T=69 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=69 . Computer: moler2 "	active (3001)
Jan 31,2018 22:03	Mar 14,2018 09:59	0.999793859643	"Matlab globalsearch"	70	"T=70 4 Qubit Piecewise pulse using fidelity with a start value at the best of T=70 . Computer: moler2 "	inactive (2938)
Jan 31,2018 22:11	Jan 31,2018 22:21	0.999777239881	"Matlab globalsearch"	70	"T=70 4 Qubit Piecewise pulse using Frobenius without squaring with a start value at the best of T=70 . Computer: moler2 "	inactive (2943)
Jan 31,2018 22:23	Jan 31,2018 22:25	0.999777239864	"Matlab globalsearch"	70	"T=70 4 Qubit Piecewise pulse using arccos(fidelity) with a start value at id=2847 . Computer: moler1 "	inactive (2950)
Feb 01,2018 08:29	Feb 01,2018 08:41	0.999777239893	"Matlab globalsearch"	70	"T=70 4 Qubit Piecewise pulse using arccos with a start value at the best of T=70 . Computer: moler2 "	inactive (2956)
Feb 01,2018 09:06	Feb 01,2018 09:06	0.9999	"Matlab globalsearch"	70	T=70 4 Qubit Piecewise pulse using fidelity at start value of zero. Computer: moler1	inactive (2960)
Mar 14,2018 12:04	Mar 14,2018 12:12	0.999777239876	"Matlab globalsearch"	70	"T=70 4 Qubit Piecewise pulse using arccos(fidelity) with a start value at id=2950 . Computer: moler1 "	active (2990)
Mar 14,2018 15:54	Apr 04,2018 10:46	0.993623299316	"Matlab globalsearch"	70	"T=70 4 Qubit Piecewise pulse using arccos with a start value at the best of T=70 . Computer: moler2 "	active (2999)
Mar 14,2018 15:55	Apr 04,2018 10:51	0.995669469756	"Matlab globalsearch"	70	"T=70 4 Qubit Piecewise pulse using Frobenius without squaring with a start value at the best of T=70 . Computer: moler2 "	active (3002)

Figure 3.9: Snapshot of the information displayed information on multiple 4-qubit instance monitored by the Optimization Database.

CHAPTER 4

APPLICATIONS

This chapter describes the applications that were solved using software packages discussed in this thesis to assist in the global optimization process. Section 4.1 describes the Quantum error correction gate design to optimize the error correction gate to minimize errors in a quantum circuit. It describes the model used in the optimization application, the method that solved the application using the Optimization Database, and the resulting pulses for the three-qubit and four-qubit cases. The second application, Rational Design of Materials in Section 4.2, describes the application to obtain the top N metastable and stable structures theoretically. This optimization is done by minimizing the crystal structure energy. This section describes the software that is used in determining the energy of the crystal structure, the global optimization algorithms used to solve the structures using the Computefarm and the Optimization Database, and the resulting structures for carbon and silicon dioxide.

4.1 Quantum error correction circuit design

Quantum computers are a promising technology that is currently being used today by D-Wave Systems [30], NASA [7], Google [23] and IBM [1]. Multiple institutes also focus on researching quantum computers and their promise to solve non-deterministic polynomial-time hard (NP-hard) problems. Prime factorization is a problem of interest in quantum computing because transistor-based computer algorithms have an exponential time complexity. Shor's algorithm is prime factorization algorithm that uses the multiple states in a quantum system. This algorithm is used in a quantum computer system to show the polynomial time complexity potential because of the use of multiple states available, unlike transistor computers that are constrained to zero or one for bit states. In a quantum computer, there are multiple states

because bits are sub-atomic particles that have multiple property values that can represent a number of states. Qubits have various properties: spin, energy level, or position in a system that can represent states. An example of using spin to represent the quantum bit is representing that states as zero (spin up) or one (spin down) similar to a transistor bit. A qubit can also have *entangled* states are in between the zero and one state, shown in Figure 4.1, because of their quantum mechanical properties.

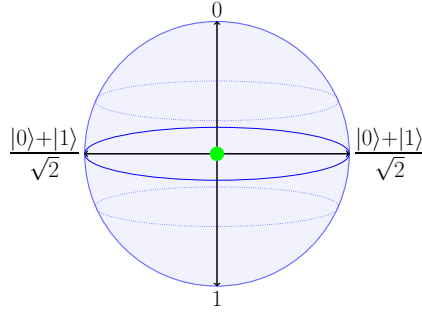


Figure 4.1: The possible spin states of a qubit.

An example of the amount of information each computer can take in is a transistor computer that has two bits can represent one of the four possible values

$$00, 10, 01, 11.$$

These combinations are represented in a transistor-based computer by using two pieces of information. In a quantum computer, two qubits can represent zero and one and any combination of states at any point in time, a process known as *superposition*. Therefore, to encode the qubits into a specific state, probability densities are given to the qubit system

$$\alpha|00\rangle, \beta|10\rangle, \gamma|01\rangle, \delta|11\rangle,$$

thus producing a linear combination

$$\alpha|00\rangle + \beta|10\rangle + \gamma|01\rangle + \delta|11\rangle.$$

Four pieces of information are given to the system, giving quantum computers the exponential advantage of 2^N (N qubits) bits of information. This advantage allows a combination

of calculation steps to be done simultaneously or in a reduced amount of steps with the use of more information by allowing each qubit to do a calculation or having multiple pieces of information readily available for one step. This extra information gives quantum computers the advantage to solve NP-hard problems in less time than transistor-based computers. Relating back to the prime factorization problem, factoring the number fifteen has been done using Shor's algorithm on a three-qubit system that took seconds to solve by Lucero [29], with Dattani and Bryans [11] factorizing the number 56153 using a four-qubit system. However, as mentioned in both papers, the error in the qubit systems prevents further work into factorization of higher numbers because the success rate becomes too low to guarantee a solution.

Error correction in quantum circuit becomes challenging because qubits can be in any given state due to their superposition ability. This unpredictable property of qubits makes it difficult to guarantee a 100% error correction rate when states are not consistent. However, methods have been developed to minimize and correct errors in a quantum computer to ensure high fault tolerance. One method that is looked at in this thesis is the controlled-Z gate that uses control-phase gates to minimize the error in the qubit system.

Three errors that occur in the control-phase gates are *decoherence*, *state leakage*, and *control time* [6]. The first error, decoherence, occurs when outside particles interact with the qubit system particles. When these particles interact, they cause a change in the energy of the qubit that further interferes with the superposition of the qubit system. This interference over time makes the qubit system lose information about its original state, thus, causing an error in the result. Certain materials can be used to minimize the error by protecting the qubit system from incoming particles to reduce interactions. A second error that can occur is state leakage; this error occurs when other states other than zero or one are measured. When other states other than zero or one are measured, the output cannot assign a value to these other state values. Thus, they are seen as noise in the measurements. Large gate times are used to avoid extra information from the measured qubit states because the qubits tend to transition to lower states, zero or one, over time. The third error is the control error; this error occurs in the gate itself, where reflections or stray inductance in wiring affects the qubit system. Qubits are controlled by frequencies that stabilize the states the

qubits are in; inductance from the wiring can disturb this control and cause an error in the system by altering the states. Low-frequency pulses known as *controlled* pulses are sent into the qubit system to stabilize the qubit states to mitigate the inductance interference. Minimizing the controlled error is done by designing a gate that generates a controlled pulse to account for the overall circuit's inductance. Experimentally, it is challenging to design a gate that stabilizes the qubit system for a given circuit because the gate has to be fine-tuned. A model of the N -qubit system is used to speed up this process to obtain a gate design for the controlled-Z phase to reduce the controlled error in a multi-gate circuit. The error correction percentage for only the modelled control error is known as *intrinsic fidelity*. The intrinsic fidelity of an error correction gate must be greater than 99.99% to guarantee a high fidelity in an experimental environment. Experimentally, it is difficult to obtain a sufficiently high fidelity because of decoherence that occurs from the setup. The upper threshold of the gate fidelity that can be obtained experimentally is 99.9% [45] when the intrinsic fidelity is optimized to 99.99%. The loss of precision in the gate fidelity is due to the other errors that are not accounted for in intrinsic fidelity.

In this application, the controlled-Z (CZ) gate is optimized to obtain an intrinsic fidelity of 99.99% for a three-qubit and four-qubit system. The qubit system uses superconducting charged qubits known as *transmons*. Transmons are used because of the reduced sensitivity to charged noise that aids in reducing error. The possible transmon energy states, j , used for this simulation are $j = \{0, 1, 2, 3\}$, where the zero and one states make up the *computation subspace*, and the other energy states, two and three, are in the *entanglement subspace*. The simulation assumes a chain of qubits, shown in Figure 4.2, where their locations in the system are represented as k .

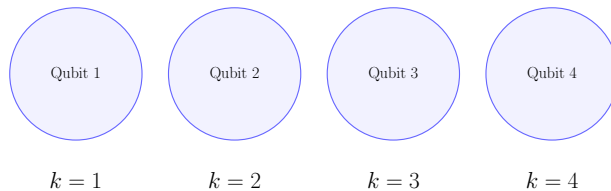


Figure 4.2: Chain of four qubits in a 4-qubit system.

Each transmon receives a pulse from the error correction gate while the gate is open for

a time, t , on the order of ns. The shifted frequencies that generate a pulse for each transmon are represented as $\Delta_k(t)$ that are bounded between

$$-2.5 \text{ MHz} \leq \Delta_k(t) \leq 2.5 \text{ MHz}. \quad (4.1)$$

The anharmonicity of the shifted frequencies of each transmon is represented as η and is set to 200 MHz for this gate design. The energy of transmon k at energy level j is

$$E_{kj} = h(j\Delta_k(t) - \eta),$$

where h is Planck's constant.

The energy transition between each transmon is represented as a Hamiltonian that is the sum of Kronecker products with the identity matrix

$$\frac{\hat{\mathbf{H}}(\Delta_k(t))}{h} = \sum_{k=1}^N \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \Delta_k(t) & 0 & 0 \\ 0 & 0 & 2\Delta_k(t) - \eta & 0 \\ 0 & 0 & 0 & 3\Delta_k(t) - \eta' \end{pmatrix}_k + \sum_{k=1}^{N-1} \frac{g_k}{2} (\mathbf{X}_k \otimes \mathbf{X}_{k+1} + \mathbf{Y}_k \otimes \mathbf{Y}_{k+1}),$$

where η' is 3η for this simulation, g_k is the coupling strength that is set to 30 MHz; each block corresponds to a fixed number of excitations that act on a Hilbert space $\mathcal{H}_4^{\otimes N}$. The \mathbf{X} and \mathbf{Y} are the coupling operators [14]

$$\mathbf{X}_k = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \sqrt{2} & 0 \\ 0 & \sqrt{2} & 0 & \sqrt{3} \\ 0 & 0 & \sqrt{3} & 0 \end{pmatrix}_k, \quad \frac{\mathbf{Y}_k}{i} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & -\sqrt{2} & 0 \\ 0 & \sqrt{2} & 0 & -\sqrt{3} \\ 0 & 0 & \sqrt{3} & 0 \end{pmatrix}_k,$$

are the generalized Pauli operators. The coupling operators describe the interaction between each transmon and its neighbour. This simulation assumes nearest-neighbour coupling, where the coupling interaction is only strong between the closest neighbour and weaker for further transmons, shown in Figure 4.3.

The Hamiltonian is then truncated up to the N excitation subspace

$$\hat{\mathbf{H}}_p(\Delta(t)) = \mathcal{O}_N \hat{\mathbf{H}}(\Delta(t)) \mathcal{O}_N^\dagger, \quad \text{where } \mathcal{O}_N^\dagger = \overline{\mathcal{O}_N^T} \quad (4.2)$$

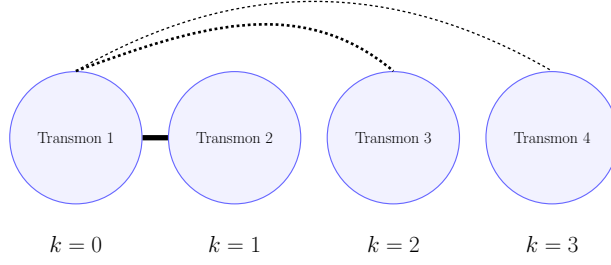


Figure 4.3: The interaction strength between the first transmon ($k = 0$) and other transmons is shown by the thickness of the lines connecting them.

because it is the highest excitation that can be achieved in the computational space. The Hamiltonian system (4.2) is then evolved over the overall gate time Θ in t time steps

$$\mathbf{U}(\Delta_k(\Theta)) = \mathcal{T} \exp \left(-i \int_0^\Theta \hat{\mathbf{H}}(\Delta_k(t)) dt \right), \quad (4.3)$$

where \mathcal{T} is the time-ordering evolution operator [12]. The unitary operator (4.3) is then projected to the computational subspace \mathcal{P} to give a 2^N matrix

$$\mathbf{U}_{\mathcal{P}}(\Delta_k(\Theta)) = \mathcal{P} \mathbf{U}(\Delta_k(\Theta)) \mathcal{P}^\dagger. \quad (4.4)$$

This matrix (4.4) is then shifted to compensate any errors in the numerical simulation by

$$\mathbf{U}_{\text{target}} = \mathbf{U}_l(\phi_1, \phi_2, \dots, \phi_k) \mathbf{U}_{\text{cz}} \mathbf{U}_r(\phi_1, \phi_2, \dots, \phi_k),$$

where the ideal CZ gate is

$$\mathbf{U}_{\text{cz}} = \begin{pmatrix} 1 & \dots & 0 & 0 \\ \vdots & \ddots & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

and the unitary phase shift on the z -axis is

$$\mathbf{U}_{l,r}(\phi_1, \phi_2, \dots, \phi_k) = \mathbf{R}_z(\phi_1) \otimes \mathbf{R}_z(\phi_2) \otimes \dots \otimes \mathbf{R}_z(\phi_k),$$

where

$$\mathbf{R}_z(\phi_k) = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\phi_k} \end{pmatrix}.$$

The phase ϕ_k of each transmon position k is the angle between the diagonal values of the projected unitary evolution (4.4).

The intrinsic fidelity is then determined by

$$\mathcal{F}(\Delta_k(\Theta)) = \frac{1}{2^N} \left| \text{Tr} \left(\mathbf{U}_{\text{target}}^\dagger \mathbf{U}_{\mathcal{P}}(\Delta_k(\Theta)) \right) \right|.$$

This model represents the objective function to optimize the frequencies, $\Delta_k(t)$, for an N -transmon system to obtain the feasible intrinsic fidelity of 99.99% for a minimal duration time Θ .

The feasibility problem is formulated as to find $\Delta_k(t)$ in the domain such that

$$0.9999 \leq \mathcal{F}(\Delta_k(t)). \quad (4.5)$$

In practice, the following optimization formulation is used

$$\min_{\Delta_k(t) \in \mathbb{D}} (1 - \mathcal{F}(\Delta_k(t))), \quad (4.6)$$

subject to (4.5), where \mathbb{D} is (4.1). In practice, the optimization formulation (4.6) is used instead because it converged to a solution faster than the feasibility formulation (4.5).

In this thesis, the three-qubit and four-qubit cases are optimized using MATLAB's global search algorithm from the global optimization toolbox [37] with a non-linear constraint to represent the feasibility condition (4.5). Other global optimization solvers (PSO, GCPSO, and DIRECT) were used; however, they did not obtain a solution. A direct method is used to minimize the overall gate time by simply solving each case with a smaller gate time for each feasible solution is attained. The Optimization Database is then used to monitor multiple gate time instances and the progress. The feasible solution pulse for the minimum gate time of 23 ns for the three-qubit case is shown in Figure 4.4.

The feasible pulse for minimal duration time of 70 ns for the four-qubit case is shown in Figure 4.5.

Obtaining a solution for the three-qubit and four-qubit case provides a gate design for multigate circuits for quantum computers. The reformulation of the problem (4.5) also allows a new method of solving quantum error control problem and potentially other various quantum problems. Currently, further work is being done on the three-qubit and four-qubit

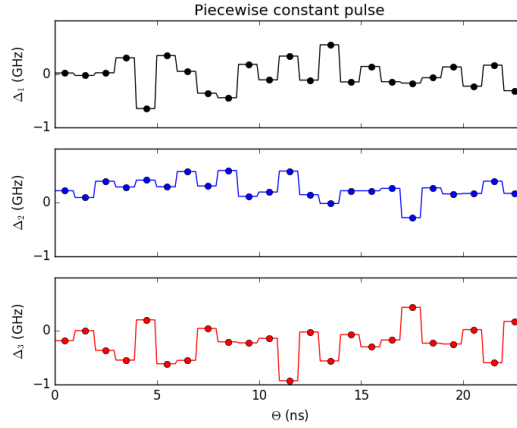


Figure 4.4: Piecewise pulse for the three-qubit case with a duration gate time of 23 ns.

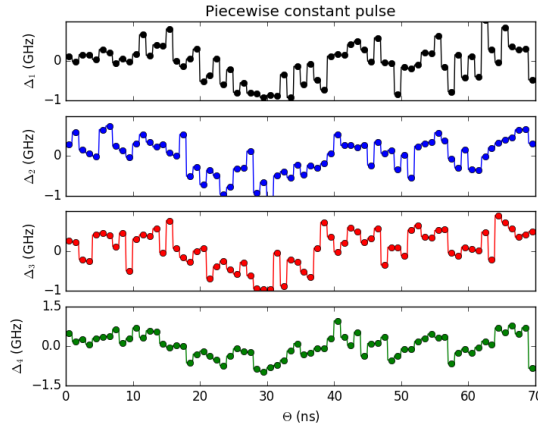


Figure 4.5: Piecewise pulse for the four-qubit case with a duration gate time of 70 ns.

to obtain a minimal gate time with a feasible solution. Global optimization simulations are being done on the five-qubit case in hope to obtain the desired intrinsic fidelity value. The current results on the three-qubit, four-qubit, and five-qubit cases are shown in Appendix C.

4.2 Crystal structure prediction

The Rational Design of Materials project uses global optimization on a given chemical compound to build and predict potential crystal structures. By predicting stable and metastable

crystal structures, new structures can be discovered, and their properties like hardness, thermal conductivity, and electrical conductivity can be calculated. Calculating these properties furthers the understanding of the crystal structure and potential applications. Another advantage of globally optimizing a compound to obtain crystal structures is that it is more cost-effective than the process of using experimentation.

Global optimization optimizes the internal structure energy that can be calculated by various software packages such as the Vienna Ab-initio Simulation Package (VASP)[24, 25]. VASP allows for the user to specify the environment (temperature, Kelvin, and pressure, Pascal) that are not possible in the experimental lab. In this application, VASP is used to calculate the enthalpy of various material compositions that indicate the structure’s stability. Enthalpy at a given pressure is the sum of the internal crystal energy and the external pV (pressure-volume) energy. The lower the enthalpy, the more stable the structure. By globally optimizing over the enthalpy, the lattice structure and atom positions of metastable and stable crystal structures are obtained. CALYPSO (Crystal Structure AnaLYsis by PSO)[43] is a software package that uses the global optimization algorithm PSO to find crystal structures from material compositions. It uses various software packages including VASP to calculate the enthalpy of a structure; CALYPSO also takes into account the symmetry of the structure and bond lengths between the atoms to speed up the optimization process.

One challenge when using global optimization algorithms to predict crystal structures is keeping track of each metastable structure observed and the properties they may have while optimizing for the stable structure. Metastable structures potentially have a multitude of various properties. For example, carbon has many metastable structures with many properties; one structure of interest is diamond because it is one of the hardest materials currently known.

The Optimization Database is used to store metastable structures during the global optimization process to help determine the top metastable structures. The algorithm used to solve carbon is GCPSO from pythOPT [42]. VASP is used in the objective function for this problem that takes in three configuration files, INCAR, POTCAR, and POSCAR, and outputs a file called OUTCAR that contains the enthalpy of the structure.

VASP locally optimizes the structure by generating a mesh in the given reciprocal lattice.

The reciprocal lattice is a set of basis vectors (\mathbf{a}^* , \mathbf{b}^* , \mathbf{c}^*) that exists in the reciprocal space (also known as *k-space*) transformed from the direct space

$$\mathbf{a}^* = \frac{\mathbf{b} \times \mathbf{c}}{V} \quad \mathbf{b}^* = \frac{\mathbf{c} \times \mathbf{a}}{V} \quad \mathbf{c}^* = \frac{\mathbf{a} \times \mathbf{b}}{V}$$

such that

$$V = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}),$$

where V is the volume of the unit cell defined by (a, b, c) . The relationship between the reciprocal lattice and the unit cell is shown in Figure 4.6.

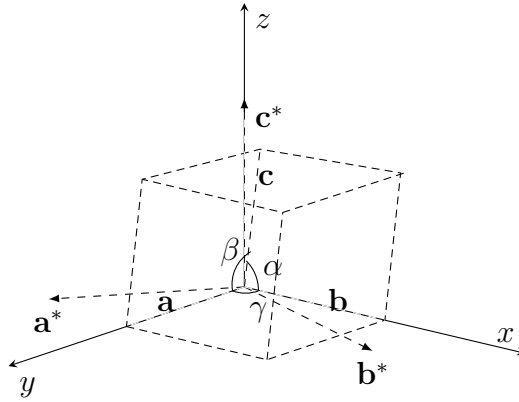


Figure 4.6: An unit cell showing the unit lengths (\mathbf{a} , \mathbf{b} , \mathbf{c}), the angles (α, β, γ), and the reciprocal lattice basis vectors (\mathbf{a}^* , \mathbf{b}^* , \mathbf{c}^*).

The reciprocal lattice is used in the generation of the POSCAR file that is utilized by VASP. VASP uses the reciprocal lattice for geometrical properties. One property that VASP uses the reciprocal lattice for is to generate the *k-mesh* from the KSPACING parameter set in the INCAR file, shown in Figure 4.7.

The *k-mesh* is made up of *k*-points that are the subdivisions of the reciprocal cell. This mesh increases the accuracy of the integration of the of the reciprocal cell, thus reducing the error in the local optimization of the enthalpy calculation. The higher the density of *k*-points, the higher the precision of the enthalpy calculation is.

The INCAR file sets the parameters for the simulation in VASP, e.g., the accuracy of the simulation, type of structure, pressure, and the number of steps to take in the optimization.

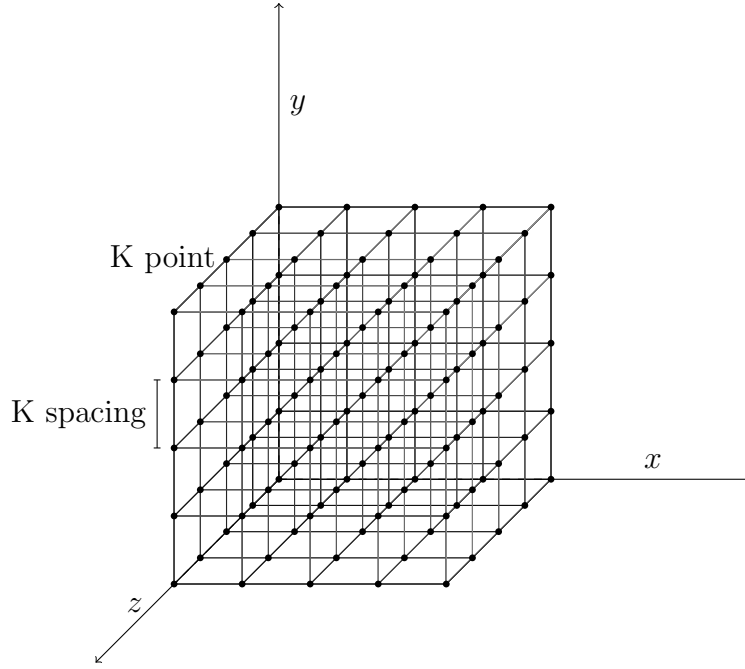


Figure 4.7: Lattice mesh based on the KSPACING value.

The format for the INCAR file is shown in Appendix D. The POTCAR file contains the pseudo potentials of ions that are comprised of simulated crystal structures. This is needed for determining the forces and enthalpy of the structure.

The OUTCAR file is the resulting output of a VASP simulation that contains the enthalpy from the local optimization of the structure. The locally optimized structure is then symmetrized to determine the space group of the structure. The space group determines whether the structure obtained is a crystal structure by analysing the symmetry. If the structure has a primitive space group known as $P1$, then it is not symmetric and not a crystal structure. In this application, the space group is determined by FINDSYM from the ISOTROPY software suite, developed by Harold Stokes from Brigham Young University [35]. The space group is stored in the Optimization Database and is used to obtain crystal structures. The identification of the space group speeds up the process of identification of metastable and stable crystal structures. The enthalpy and structure are also stored in the Optimization Database and returned to the GCPSO solver. An advantage of the GCPSO solver in pythOPT allows particle positions to be updated by the objective function. This feature of GCPSO allows VASP's locally optimized lattice, which is the decision vector, to be passed into GCPSO and

be used a decision vector in the solver. The search space on the lattice structure is provided in Table D.1.

The two structures looked at in this application is carbon and silicon dioxide. An eight-atom carbon simulation in VASP using four processors on an AMD Opteron(TM) Processor 6276 machine takes on average five to fifteen minutes to complete depending on the INCAR settings. A twelve-atom silicon dioxide simulation in VASP using four processors on the same machine takes on average three hours to complete. To minimize the computational time and computer resources on a single machine, Computefarm is used on the diatomic structure silicon dioxide with GCPSO to obtain the metastable and stable structures.

	Variable	Range
Unit cell	α	$(80^\circ, 130^\circ)$
	β	$(80^\circ, 130^\circ)$
	γ	$(80^\circ, 130^\circ)$
	a	$(2\text{\AA}, 4\text{\AA})$
	b	$(2\text{\AA}, 4\text{\AA})$
	c	$(2\text{\AA}, 4\text{\AA})$
Atom _{<i>n</i>}	x_n	$(0, 1)$
	y_n	$(0, 1)$
	z_n	$(0, 1)$

Table 4.1: Search space for Rational Design of Materials project for an n -atom system.

The speed up for one iteration of GCPSO using four processors with VASP is shown in Table 4.2.

Four processors were allocated to VASP in this application because of the processor constraint on the client machines (Intel(R) core(TM) i7-6700 CPU) used with Computefarm.

By using the Optimization Database, the ten lowest enthalpy structures for carbon are shown in Figure 4.8. The lowest enthalpy structure in Figure 4.8 is graphite, shown in Figure 4.9, and a metastable structure, diamond, is shown in Figure 4.10.

The five minimal enthalpy structures for silicon dioxide are shown in Figure 4.11. The

Method	Average time (seconds) for 1 objective function evaluation	Average time (seconds) for 1 iteration (20 objective function evaluations)
GCPSO	11970.486	239401.356
GCPSO with ComputeFarm	11970.486	12131.577

Table 4.2: Time comparisons between GCPSO and GCPSO with ComputeFarm for silicon dioxide.

lowest enthalpy structure of silicon dioxide in Figure 4.11 is cristobalite, shown in Figure 4.12, and a metastable structure of silicon dioxide, stishovite, is shown in Figure 4.13.

4.3 Concluding remarks

By using ComputeFarm and the Optimization Database, solutions for both applications were obtained. In Section 4.1, a feasible pulse for the three-qubit and four-qubit case is obtained using the Optimization Database. In Section 4.2 used ComputeFarm and the Optimization Database to obtain various metastable and stable structures for carbon and silicon dioxide. Both software packages provide an advantage to obtaining solutions faster through distribution of tasks and monitoring of the global optimization process.

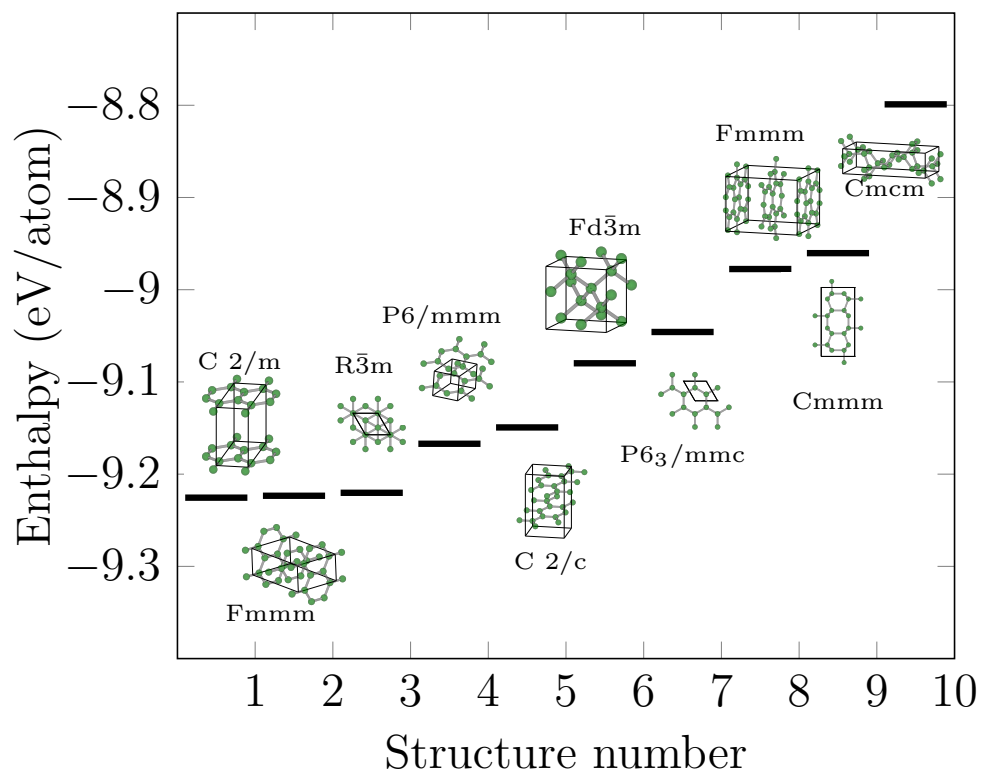


Figure 4.8: Ten minimal enthalpy structures of carbon.

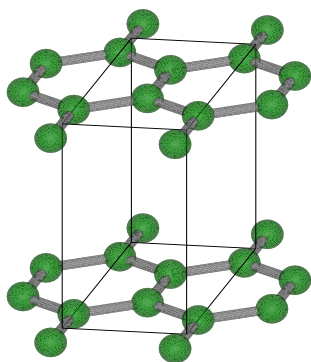


Figure 4.9: Graphite (C 2/m).

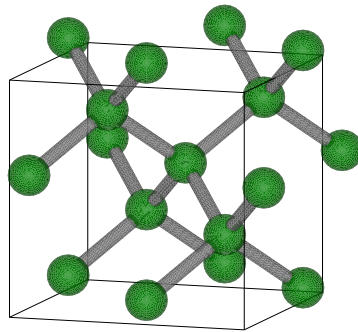


Figure 4.10: Cubic-diamond, ($Fd\bar{3}m$).

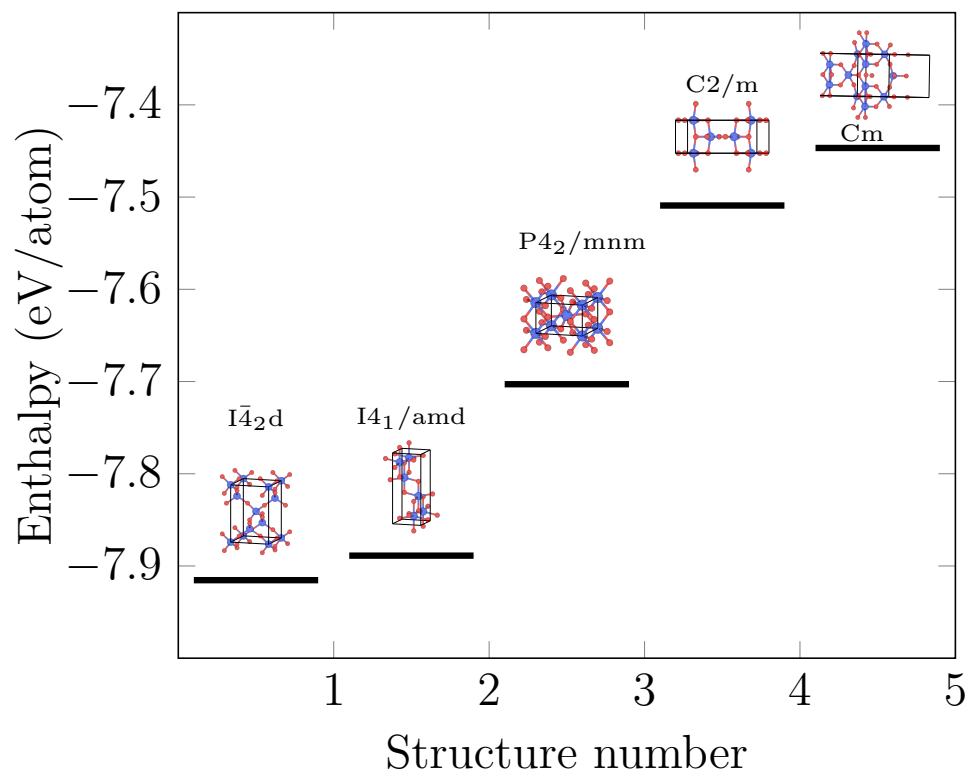


Figure 4.11: Five minimal enthalpy structures of silicon dioxide.

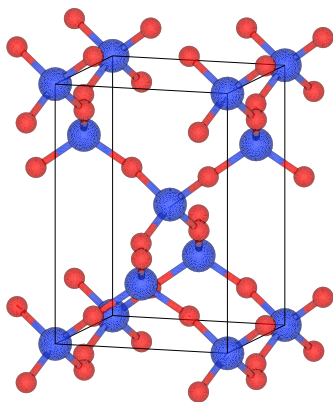


Figure 4.12: Cristobalite ($I\bar{4}_2d$).

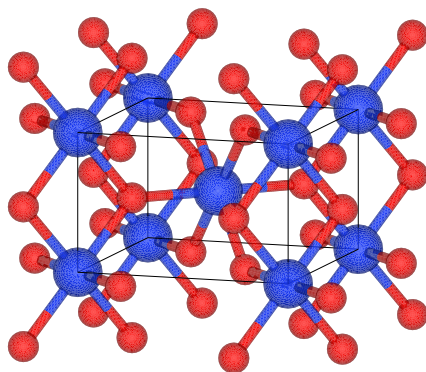


Figure 4.13: Stishovite ($P4_2mm$).

CHAPTER 5

CONCLUSIONS AND SUGGESTIONS FOR FUTURE RE- SEARCH

In this thesis, global optimization methods were used in combination with software packages to solve two applications. The two software packages developed for this thesis is the Optimization Database and Computefarm. They developed to aid in the process of optimizing the two applications: Quantum error correction gate design and Rational Design of Materials.

5.1 Conclusions

The conclusions are divided up as follows: an overview of the software, quantum error correction gate design, and Rational Design of Materials.

5.1.1 Overview of the software

In this thesis, two software packages were developed to be used with global optimization solvers to assist the optimization process. The first software package discussed is Computefarm; this software package utilizes distributed local computer resources to run objective function evaluations in parallel. By doing this, the iteration step time is reduced and the overall optimization time is decreased. Computefarm's API is shown in Appendix A and is used in solving the problem of finding the top metastable structures of silicon dioxide; see Section 4.2.

The second software packaged that is developed for this thesis is the Optimization Database. This software package stores data from a global optimization process and extra information

from the objective function. An example of extra information to be stored is the symmetry of a crystal structure for post-processing. Another advantage of this software is the built-in monitoring systems to allow the user to see the progress of the global optimization process. Two monitoring schemes that were developed are:

- email notifications
- online monitoring

The email notifications and online monitoring (shown in Figure 3.6 and Figure 3.9) were used in the quantum error correction gate design to assist in the brute force search to obtain the minimal gate time. The monitoring schemes aided in knowing a feasible solution of an intrinsic fidelity of 99.99% is obtained and when various global optimization algorithms were not performing well. The API for the Optimization Database is shown in Appendix B.

Both software packages were used in combination with a global optimization algorithm to assist in the process of solving the two applications.

5.1.2 Quantum error correction gate design

In previous work on quantum error correction gate design, a minimum gate time of 26 ns is obtained for the 3-qubit system [44] and no results obtained for the 4-qubit system. In this thesis, a lower gate time is obtained for the 3-qubit system and a solution is obtained for the 4-qubit system. Both systems were optimized by *MATLAB*'s Global Search algorithm with the feasibility formulation (4.5). Other global optimization algorithms were also used in the attempt to obtain results for these problems; however, *MATLAB*'s Global Search algorithm outperformed the other algorithms by obtaining the high fidelity value of 99.99% for both systems. In combination with the global optimization algorithm, the Optimization Database was used to monitor the optimization process and to store the results of the optimizations. By using the Optimization database a brute force method of decreasing the gate time for each instance of the three-qubit system problems is done to obtain the gate times of 23 ns for the 3-qubit problem (see Figure 4.4) and 70 ns for the 4-qubit problem (see Figure 4.5). Obtaining these results further pushes the error correction design for higher multi-gate qubit systems that can be used in future circuit designs for quantum computers. Another finding

from this application is the new reformulation of the problem that could solve other quantum error correction problems.

5.1.3 Rational Design of Materials

Prediction of crystal structures from a known compound is a well known global optimization problem in material science. Various software, like VASP [26] and CALYPSO [43], are used to predict crystal structure that is not cost effective for experimentation. However, global optimization solvers including CALYPSO only return a single stable structure as the best solution. This does not address the interest in metastable states; for example the metastable structure diamond, shown in Figure 4.10. In this thesis, the collection of metastable structures is stored using the Optimization Database by storing the extra information of the symmetry of the crystal structure. By storing this information in a database, a simple query to obtain the symmetrical top low energy states. In combination with GCPSO and Computefarm, the optimization process obtained the top ten crystal structures for carbon, shown in Figure 4.8, and the top five crystal structures of silicon dioxide, shown in Figure 4.11. The usage of Computefarm reduces the overall optimization time by parallelizing the objective function evaluations by a distributed system. Then structures with symmetry were stored along with their energy value as an objective function result to be used in later processing.

5.2 Suggestions for future work

There are several ways in which the work presented can be further expanded.

5.2.1 Software packages

The software presented in this thesis is being continuously improved upon and utilized in solving other complex applications. Computefarm currently only works with available network ports on a Linux operating system. One improvement to Computefarm's cross-compatibility is to implement a web-based server system to allow connections from other operating systems of local machines. The web-based server can then send out objective function evaluations

to various clients and use a similar database scheme and monitoring system to monitor the connections with various clients. This monitoring scheme can allow for further analysis Computefarm. Another software improvement can include the additional implementation of using the socket library *ZeroMQ* [16], a socket framework for distributive applications. These improvements can lead to performance analysis on Computefarm in comparison to other open-source software packages, like BOINC.

The Optimization Database is currently only using a relational database for the global optimization process. This schema can be expanded to include more information on the global optimization solver to monitor termination conditions and various variables that may change throughout the process for further analysis. The database flexibility can be improved by using an object-oriented database, for example, *NoSQL* [38]. This flexibility will allow storage of objects pertaining to the global optimization process as well as information stored in the objective function.

5.2.2 Quantum computer error correction gate design

The quantum error correction gate design is an ongoing research field with the goal of building large qubit systems. The quantum error correction gate design application can expand into optimizing other gate designs or solving various other quantum error problems. Currently, the search for a minimum gate time for the three-qubit, four-qubit, and five-qubit is ongoing. Another interest in expanding this application is to other techniques and software to solve the feasible formulation (4.5) of the problem and to do analysis on the various methods. This can help understand the properties and characteristics of quantum error correction problems to further solve the problems in that research field.

5.2.3 Rational Design of Materials

Rational Design of Materials is continuously optimizing other diatomic compositions; this includes the use of diffraction grating patterns to optimize for metastable structures. Other future work for this project includes determining crystal structure properties from the obtained metastable and stable structures to potentially find new structures and using experi-

mental data, like diffraction patterns, to determine new metastable crystal structures. Other interests of the software environment developed for this project is to provide a graphical display of the metastable structures.

REFERENCES

- [1] IBM universal quantum computing. <https://www.research.ibm.com/ibm-q/learn/>. Accessed: 2017-09-29.
- [2] G. Accary, D. Morvan, and S. Me. The Human Line-1 Retrotransposons Creates DNA Double Strand Breaks. *Fire Safety Journal*, 93(5):173–178, 2008.
- [3] C. S. Adjiman. A global optimization method, aBB, for general twice-differentiable constrained NLPs. *Journal of Chemical Information and Modeling*, 53(9):1689–1699, 2013.
- [4] H. Aguiar and O. Junior. *Evolutionary Global Optimization , Manifolds and Applications*. Springer, 2016.
- [5] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.
- [6] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O’Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and J. M. Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.
- [7] M. Benedetti, J. Realpe-Gómez, R. Biswas, and A. Perdomo-Ortiz. Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Physical Review A*, 94(2):022308, 2016.
- [8] U. Can and B. Alatas. Physics Based Metaheuristic Algorithms for Global Optimization. *American Journal of Information Science and Computer Engineering*, 1(3):94–106, 2015.
- [9] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *1999 ICEC, Washington DC*, pages 1951–1957, 1999.
- [10] A. R. Conn, N. I. Gould, and P. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM journal on numerical analysis*, 25(2):433–460, 1988.
- [11] N. S. Dattani, N. Bryans, E. Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariani, A. Megrant, P. O. Malley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland, and J. M. Martinis. Quantum factorization of 56153 with only 4 qubits. *Nature Physics*, 8(143):1–6, 2014.

- [12] G. Dattoli. Time-ordering techniques and solution of differential difference equation appearing in quantum optics. *Journal of Mathematical Physics*, 72(3):772–780, 1986.
- [13] T. Desell. *Asynchronous global optimization for massive-scale computing*. PhD thesis, Rensselaer Polytechnic Institute, 2009.
- [14] J. Ghosh, A. Galiutdinov, Z. Zhou, A. N. Korotkov, J. M. Martinis, and M. R. Geller. High-fidelity controlled- σ_Z gate for resonator-based superconducting quantum computers. *Physical Review A - Atomic, Molecular, and Optical Physics*, 87(2):1–19, 2013.
- [15] F. Glover. A Template for Scatter Search and Path Relinking. *Artificial Evolution*, 1363(February 1998):2–51, 1998.
- [16] P. Hintjens. Zeromq: The guide. URL <http://zeromq.org>, 2010.
- [17] R. Hooke and T. A. Jeeves. “Direct Search” Solution of Numerical and Statistical Problems. *Journal of the ACM*, 8(2):212–229, 1961.
- [18] Y. Hung and W. Wang. Accelerating parallel particle swarm optimization via GPU. *Optimization Methods and Software*, 27(1):33–51, 2012.
- [19] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.
- [20] A. Kaveh. *Advances in Metaheuristic Algorithms for Optimal Design of Structures*. Springer, 2014.
- [21] J. Kennedy and R. Eberhart. Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4:1942–1948, 1995.
- [22] J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, 5:4104–4108, 1997.
- [23] I. D. Kivlichan, N. Wiebe, R. Babbush, and A. Aspuru-Guzik. Bounding the costs of quantum simulation of many-body physics in real space. *Journal of Physics A: Mathematical and Theoretical*, 50:305301, 2017.
- [24] G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, 6(1):15–50, 1996.
- [25] G. Kresse and J. Furthmüller. Efficient iterative schemes for *ab initio* total-energy calculations using a plane-wave basis set. *Physical Review B*, 54(16):11169–11186, 1996.
- [26] G. Kresse and J. Furthmüller. Vienna ab-initio simulation package (vasp). *Vienna: Vienna University*, 2001.

- [27] T. Krink, J. S. Vesterstrom, and J. Riget. Particle swarm optimisation with spatial particle extension. *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, 2:1474–1479, 2002.
- [28] L. Liberti. Introduction to global optimization. *Ecole Polytechnique*, 2000.
- [29] E. Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. O. Malley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland, and J. M. Martinis. Computing prime factors with a Josephson phase qubit quantum processor. *Nature Physics*, 8(10):719–723, 2012.
- [30] F. Neukart, G. Compostella, C. Seidel, D. von Dollen, S. Yarkoni, and B. Parney. Traffic flow optimization using a quantum annealer. *arXiv preprint arXiv:1708.01625v2*, pages 1–12, 2017.
- [31] J. D. Pintér. Global Optimization: Software, Test Problems, and Applications. *Handbook of Global optimization*, 2:515–569, 2002.
- [32] G. S. Sadasivam and D. Selvaraj. A novel parallel hybrid pso-ga using mapreduce to schedule jobs in hadoop data grids. In *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pages 377–382, Dec 2010.
- [33] Y. Shi and R. Eberhart. A Modified Particle Swarm Optimizer. *IEEE International Conference*, pages 69–73, 1998.
- [34] D. Simon. *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.
- [35] H. T. Stokes and D. M. Hatch. FINDSYM: program for identifying the space-group symmetry of a crystal. *Journal of Applied Crystallography*, pages 237–238, 2005.
- [36] R. G. Strongin and Y. D. Sergeyev. *Global optimization with non-convex constraints: Sequential and parallel algorithms*, volume 45. Springer Science & Business Media, 2013.
- [37] Users, MathWorks Global Optimization Toolbox. Guide (R2017b). *MATLAB Global Optimization Toolbox Users Guide*, 2017.
- [38] S. Tiwari. *Professional NoSQL*. John Wiley & Sons, 2011.
- [39] V. Torczon. Pattern search methods for nonlinear optimization. In *SIAG/OPT Views and News*. Citeseer, 1995.
- [40] F. Van den Bergh and A. Engelbrecht. A new locally convergent particle swarm optimizer. *IEEE conference on systems, man and cybernetics*, 2002.
- [41] M. Veltman. Algebraic Techniques. *Computer Physics Communication* 3, pages 75–78, 1972.
- [42] K. M. Voss. *pythOPT: A problem-solving environment for optimization methods*. PhD thesis, University of Saskatchewan, 2017.

- [43] Y. Wang, J. Lv, L. Zhu, and Y. Ma. CALYPSO: A method for crystal structure prediction. *Computer Physics Communications*, 183:20632070, 2012.
- [44] E. Zahedinejad, J. Ghosh, and B. C. Sanders. High-Fidelity Single-Shot Toffoli Gate via Quantum Control. *Physical Review Letters*, 114(20):200502, 2015.
- [45] E. Zahedinejad, S. Schirmer, and B. C. Sanders. Evolutionary algorithms for hard quantum control. *Physical Review A - Atomic, Molecular, and Optical Physics*, 90(3):1–10, 2014.

APPENDIX A

SOFTWARE APPLICATION INTERFACE

A.0.1 Computefarm interface

The following snippets of code show the implementation of Computefarm in pythOPT's PSO algorithm. The first snippet of code shows the initialization of Computefarm that requires a port number the client machines are listening on and the script name of the objective function. The termination method is also called at the end of the evolution step. This method gracefully closes all connections and kills the threads being used in Computefarm.

```
def _evolution(self):
    """ Perform evolution of the swarm

    The main loop of the algorithm
    """

    import signal

    def sigint_handler(signal, frame):
        raise StoppingCriterion()
    signal.signal(signal.SIGINT, sigint_handler)

    self.state.cur_evaluations = 0
    self.state.cur_iterations = 0
    try:
        # Setup Computefarm
        CF.initialize_computefarm(self.settings['port'],
                                   self.settings['script_name'])
    except:
        raise ValueError("Error: _Port_or_script_name_was_not"
                        "_set_properly_please_look_over_you_settings_dictionary")
    self._initial_evolution()
    self._update_answer()
    self._initial_global_best = self.state.minimum
    self._check_termination()

    while True:
        self.update_weights()
        self._evolve_particles()
        self._update_answer()

        self.state.cur_evaluations += len(self.particles)
        self.state.cur_iterations += 1
```

```

        self._check_termination()
        # Terminate ComputeFarm software
    CF.termination()

```

The second snippet of code shows the usage of ComputeFarm to evaluate a list of particles that contain decision vectors. The objective function evaluations are stored in a list and returned back to the PSO algorithm.

```

def _initial_evolution(self):
    """ initial step for evolution
    """

    # we need global best x so explicit evaluation first
    #map(methodcaller('evaluate'), self.particles)
    #self.state.cur_evaluations += len(self.particles)
    position = []
    for particle in self.particles:
        position.append(particle.x)
    positions = np.array(position)
    # Call ComputeFarm to evaluate the particle list
    results = CF.computeFarm(positions)

    for (particle, result) in itertools.izip(self.particles, results):
        # Check if Optimization Database is set
        if self.db:
            self.db.save(-1*result,
                        np.array(particle.x),
                        False,
                        self.db_settings['extra_values'])
        particle.current_value = result
        if result < particle.bestx_value:
            particle.bestx = np.array(particle.x)
            particle.bestx_value = result
            print particle.bestx_value
            update_info = (particle.bestx_value, particle.bestx)
            particle.notify_observers(update_info)

```

The third snippet of code shows the continuation of ComputeFarm being used in the evolution step in the PSO algorithm.

```

def _evolve_particles(self):
    """ Evolve the swarm once
    """

    position = []
    for particle in self.particles:
        particle._update_velocity()
        particle._update_position()

```

```

        position.append(particle.x)
    positions = np.array(position)
    # Call ComputeFarm to evaluate particle list
    results = CF.computeFarm(positions)

    for (particle,result) in itertools.izip(self.particles,result):
    if self.db:
        print "saving_data..."
        self.db.save(-1*result,
                    np.array(particle.x),
                    False,
                    self.db_settings['extra_values'])
    print "update_particle_info"
    particle.current_value = result
    if result < particle.bestx_value:
        particle.bestx = np.array(particle.x)
        particle.bestx_value = result
        print particle.bestx_value
        update_info = (particle.bestx_value,particle.bestx)
        particle.notify_observers(update_info)

```

ComputeFarm PSO example The following code shows an example of calling the ComputeFarm PSO (CFPSO) algorithm in pythOPT.

```

from src import Vasp, PythoptSolvers, Database
from collections import OrderedDict as Odict
import numpy as np

```

```

space = [(2, 7)]*3+[(70, 130)]*3+[(0, 1)]*(3*4)+[(0, 1)]*(3*8)
data_size = len(space)

```

```

database_name = 'simlab_db'
user = 'simlabuser1'
host = 'db.cs.usask.ca'
password = 'simlab_01'

```

```

settings = {
    'eval_lim': 10000000,
    'nparticles': 50,
    'seed': 10,
    'max_noimprov': 60,
    'method': "'GCPSO'",
    'dimension': data_size,
    'lb': "'[2,50,0]'",
    'ub': "'[7,110,1]'",
}

```

Optimization settings	Value
Total number of objective function evaluations	30
Seed	300
Total number of particles	30
Total number of client machines potentially available	35
Objective function evaluation time	1959 seconds
Main Computer Specifications	Value
CPU	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Number of processors	8
RAM	64 GB
Client Computer Specification	Value
CPU	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Number of processors	8
RAM	16 GB

Table A.1: Global optimization and computer specifications used for the Computefarm performance experiment.

```

'note': "'SiO2 , _pressure_0_GPa'",
'extra_fields': {'symmetry': 'text'},
'table_name': 'sio',}

crystal_structure = Odict()
crystal_structure['potcar'] = 'SiO'
crystal_structure['pressure'] = 0
crystal_structure['Si'] = 4
crystal_structure['O'] = 8

db = Database(settings , database_name , user , host , password)
db.setup()

vasp_settings = {'E': -70.0,
                 'd': 1.18}

vasp = Vasp(crystal_structure , vasp_settings , db)

pythopt = PythoptSolvers(space , settings , vasp)
pythopt._GCPSO()

```

A.0.2 Performance test specifications

APPENDIX B

OPTIMIZATION DATABASE

B.0.1 Optimization Database interface

The Optimization Database class Database() requires the following information:

- settings dictionary
- database name
- database user name
- database host name
- database password

The settings dictionary options are:

- dimensions of the decision vector - integer
- name of the optimization method - string
- upper bound of the domain - string
- lower bound of the domain - string
- table name into which to insert the results
- extra fields - dictionary where the key is the column name and value is the string name of the type
- objection function evaluation starting value - large integer
- best objective function evaluation value - large integer
- note describing the problem and instance - string

The setup method requires no arguments and creates the database tables and inserts the Optimization Database information passed into the class.

The save method arguments are:

- objective function evaluation - float
- decision vector - float array
- minimum flag - boolean such that best minimum value is stored if true; otherwise store the best maximum value

- extra information on the problem - dictionary such that the key is the column name and the value is the value to be stored in that column

The `get_best_value` method takes in a condition described by a dictionary such that the key is column for the condition and the value is a conditional value. This method then returns a dictionary contain the best objective function evaluation value satisfying the condition and the corresponding decision vector.

The following are examples of the use cases of the Optimization Database in MATLAB and python.

MATLAB The first snippet of code show the setup of the Optimization Database class.

```
try
    db_lib = py.importlib.import_module('database');
catch
    warning('Python_database_is_not_currently_in
    .....this_folder_or_the_software_cannot_find_it..')
    prompt = 'Would_you_like_to_continue_anyways
    .....without_a_database?(yes_or_no)';
    response = input(prompt);
    if response == 'no'
        display('Terminating_Nqubit_code,_have_a_nice_day!');
        exit;
    end
end
extra_field = py.dict(pyargs('T','int'));
db_settings = py.dict(pyargs('dimension',QubitInfo.n*QubitInfo.T,
                             'method','Matlab_globalsearch','ub',u
                             'lb',lb,'table_name',tablename,
                             'extra_fields',extra_field,
                             'f_value',-999,
                             'best_value',-999,'note',note));

%try
    QubitInfo.db = db_lib.Database(db_settings,db_name,user,host,passw
    QubitInfo.db.setup()
```

The second snippet of code shows an example of saving the objective function data into the Optimization Database.

```
if QubitInfo.databaseOpt == 1
    extra = py.dict(pyargs('T',QubitInfo.T));
    try
        QubitInfo.db.save(fidelity,mat2str(QubitInfo.x),
                           py.bool(false),extra);
    catch
        warning('The_following_fidelity_%f_was_not_saved_in_the_database.
        .....please_take_a_look',fidelity);
```

```

    end
end

```

The third snippet of code shows an example of obtaining the best objective function decision vector for a given primary ID value.

```

extra_field = py.dict(pyargs('T','int'));
db_lib      = py.importlib.import_module('database');

db_settings = py.dict(pyargs('dimension',QubitInfo.n*QubitInfo.T,
    'method','"Matlab_globalsearch"', 'ub',ub,'lb',lb,'table_name',
    tablename,'extra_fields',extra_field,'f_value',-999,
    'best_value',-999,'note','',));
%try
    QubitInfo.db = db_lib.Database(db_settings,
                                   db_name,
                                   user,
                                   host,
                                   password);

    if (QubitInfo.startValue == 1 )
        condition = py.dict(pyargs('T',QubitInfo.T));
    elseif (QubitInfo.startValue == 6 || QubitInfo.startValue == 7)
        condition = py.dict(pyargs('T',QubitInfo.start_T));
    else
        condition = py.dict(pyargs('id',QubitInfo.id));
    end
    best_val = QubitInfo.db.get_best_value(condition);
    best_val{'f'}
    QubitInfo.x0 = double(py.array.array('d',best_val{'x'}));

```

Python The first snippet of code shows the Optimization Database setup in python with pythOPT.

```

from src import Vasp, PythoptSolvers, Database
from collections import OrderedDict as Odict
import numpy as np

```

```

space = [(2, 7)]*3+[(70, 130)]*3+[(0, 1)]*(3*4)+[(0, 1)]*(3*8)
data_size = len(space)

```

```

database_name = 'simlab_db'
user = 'simlabuser1'
host = 'db.cs.usask.ca'
password = 'simlab_01'

```

```

settings = {
    'eval_lim': 10000000,
    'nparticles': 50,
    'seed': 10,
    'max_noimprov': 60,
    'method': "'GCPSO'",
    'dimension': data_size,
    'lb': "'[2,50,0]'",
    'ub': "'[7,110,1]'",
    'note': "'SiO2, _pressure_0_GPa'",
    'extra_fields': {'symmetry': 'text'},
    'table_name': 'sio',}

crystal_structure = Odict()
crystal_structure['potcar'] = 'SiO'
crystal_structure['pressure'] = 0
crystal_structure['Si'] = 4
crystal_structure['O'] = 8

db = Database(settings, database_name, user, host, password)
db.setup()

vasp_settings = {'E': -70.0,
                 'd': 1.18}

vasp = Vasp(crystal_structure, vasp_settings, db)

pythopt = PythoptSolvers(space, settings, vasp)
pythopt._GCPSO()

```

The second snippet of code shows the Optimization Database save method implemented in the PSO algorithm.

The third snippet of code shows the Optimization Database save method implemented in a python objective function.

B.0.2 Email interface

The following script is an example template on setup an emailing script to get periodic notifications from the Optimization Database.

```
"""
```

*Need to import the file class, to do this set your
PYTHONPATH environment variable to include the opt_Email.py path.*

Example:

*export PYTHONPATH=\$HOME/svn/Optimization_database/scripts/:\$PYTHONPATH
Suggested remark is to place this in your .bashrc*


```
#notification time for every day the script is running
#Note: suggested to run this script in a tmux
#or screen session to ensure persistant running that is need for this method.
optEmail.notificationEmail()
```

B.0.3 Website interface

The Following script is an example of calling the Optimization Database PHP script to obtain a table format for web-page containing the contents of the database table. The database function takes in three arguments:

- table name of where the data is stored
- minimum (min) or maximum (max) string indicating what is best objection function evaluation to obtain
- column to sort the data by in the table format showed on the web-page

```
<div id="3qubit" class="tabcontent">
  <h3>3-Qubit</h3>
  <?php
  database('t_3qubit','max','t');
  ?>
</div>
```

APPENDIX C

QUANTUM ERROR CORRECTION GATE RESULTS

Duration time Θ	Fidelity
26	0.999994916655
25	0.999704618107
23	0.999900531541
22	0.99978612226

Table C.1: Duration time results for intrinsic fidelity for the three-qubit case.

Duration time Θ	Fidelity
70	0.9999
69	0.999158236756
65	0.999417006969
64	0.99970883131
63	0.999808699244
62	0.999669253826

Table C.2: Duration time results for intrinsic fidelity for the four-qubit case.

Duration time Θ	Fidelity
100	0.998843049933
90	0.748563826084

Table C.3: Duration time results for intrinsic fidelity for the five-qubit case.

APPENDIX D

RATIONAL DESIGN OF MATERIALS FILES

	Variable	Range
Unit cell	α	(80° , 130°)
	β	(80° , 130°)
	γ	(80° , 130°)
	a	(2Å, 4Å)
	b	(2Å, 4Å)
	c	(2Å, 4Å)
Atom _{n}	x_n	(0, 1)
	y_n	(0, 1)
	z_n	(0, 1)

Table D.1: Search space for Rational Design of Materials project for an n -atom system.

GCPSO settings

Python script to run the code Example script to run an eight atom carbon simulation using the Rational Design software with pyhOPT.

```
from src import Vasp, PythoptSolvers, Database
from collections import OrderedDict as Odict
import numpy as np
```

```
"""
```

```
space = [a, b, c, alpha, beta, gamma, (x, y, z)*num of atoms]
"""
```

```
space = [(2, 7)]*3+[(50, 140)]*3+[(0, 1)]*(3*8)
data_size = len(space)
```

```
'''
```

```
Database settings
```

```
'''
```

```
database_name = 'simlab_db'
user = 'simlabuser1'
host = 'db.cs.usask.ca'
password = 'simlab_01'
settings = {
    'eval_lim': 10000000,
```

```

    'nparticles': 50,
    'seed': 10,
    'max_noimprov': 60,
    'method': "'GCPSO'",
    'dimension': data_size,
    'lb': "'[2,50,0]'",
    'ub': "'[7,140,1]'",
    'note': "'C8, pressure at 0 GPa'",
    'extra_fields': {'symmetry': 'text'},
    'table_name': 'C'}

crystal_structure = Odict()
crystal_structure['pressure'] = 0
crystal_structure['potcar'] = 'C'
crystal_structure['C'] = 8

vasp_settings = {'E': -70.0, \
                 'd': 1.0}

db = Database(settings, database_name, user, host, password)
db.setup()

vasp = Vasp(crystal_structure, vasp_settings, db)

pythopt = PythoptSolvers(space, settings, vasp)
pythopt._GCPSO()

```

INCAR

```

SYSTEM = local optimization
PREC = Normal
ENCUT = 520
EDIFF = 1e-05
IBRION = 2
ISIF = 3
NSW = 100
ISMear = 1; SIGMA = 0.2
POTIM = 0.10
#WaveFunction and charge
LWAVE = .FALSE.
LCHARGE = .FALSE.
#Target Pressure
PSTRESS = 0
#Finer optimization
EDIFFG = -0.5e-04

```



```

LPLANE = .TRUE.
NCORE = 4
LSCALU = .FALSE.
KSPACING = 0.3
KGAMMA = .TRUE.

```

Diamond CIF file

FINDSYM, Version 4.3.1, December 2015
 Written by Harold T. Stokes, Branton J. Campbell, and Dorian M. Hatch
 Brigham Young University

```

C_8_ crsytal
Tolerance:      0.10000
Lattice parameters, a,b,c,alpha,beta,gamma:
    2.49948    5.56447    4.94050 131.29231  60.92307 101.95492
Centering: P
Number of atoms in unit cell:
    8
Type of each atom:
8*C
Position of each atom (dimensionless coordinates)
    1    0.15789    0.07112    0.49735
    2    0.03517    0.44531    0.31104
    3    0.65789    0.57112    0.74735
    4   -0.46483    0.94531    0.56104
    5    0.65789    0.57112   -0.75265
    6    0.53517   -0.05469    0.06104
    7    0.03517    0.44531    0.81104
    8    0.15789    0.07112   -0.00265

```

```

-----
Space Group 227  Oh-7      Fd-3m
Origin at      0.60480    0.24963    0.62518
Vectors a,b,c:
    0.50000    0.50000    0.75000
   -1.50000    0.50000    0.75000
   -0.50000   -0.50000    0.25000
Values of a,b,c,alpha,beta,gamma:
    3.54252    3.54252    3.54252 90.00000  90.00000  90.00000
Atomic positions in terms of a,b,c:
Wyckoff position b
    1    0.62500    0.62500    0.62500
    2    0.37500    0.37500    0.37500

```

```

# CIF file
# This file was generated by FINDSYM
# Harold T. Stokes, Branton J. Campbell, Dorian M. Hatch
# Brigham Young University, Provo, Utah, USA

data_findsym-output
_audit_creation_method FINDSYM

_symmetry_space_group_name_H-M "F 41/d -3 2/m (origin choice 2)"
_symmetry_Int_Tables_number 227

_cell_length_a    3.54252
_cell_length_b    3.54252
_cell_length_c    3.54252
_cell_angle_alpha 90.00000
_cell_angle_beta  90.00000
_cell_angle_gamma 90.00000

loop_
_space_group_symop_id
_space_group_symop_operation_xyz
1 x,y,z
2 x,-y+1/4,-z+1/4
3 -x+1/4,y,-z+1/4
4 -x+1/4,-y+1/4,z
5 y,z,x
6 y,-z+1/4,-x+1/4
7 -y+1/4,z,-x+1/4
8 -y+1/4,-z+1/4,x
9 z,x,y
10 z,-x+1/4,-y+1/4
11 -z+1/4,x,-y+1/4
12 -z+1/4,-x+1/4,y
13 -y,-x,-z
14 -y,x+1/4,z+1/4
15 y+1/4,-x,z+1/4
16 y+1/4,x+1/4,-z
17 -x,-z,-y
18 -x,z+1/4,y+1/4
19 x+1/4,-z,y+1/4
20 x+1/4,z+1/4,-y
21 -z,-y,-x
22 -z,y+1/4,x+1/4
23 z+1/4,-y,x+1/4
24 z+1/4,y+1/4,-x

```

25 $-x, -y, -z$
 26 $-x, y+1/4, z+1/4$
 27 $x+1/4, -y, z+1/4$
 28 $x+1/4, y+1/4, -z$
 29 $-y, -z, -x$
 30 $-y, z+1/4, x+1/4$
 31 $y+1/4, -z, x+1/4$
 32 $y+1/4, z+1/4, -x$
 33 $-z, -x, -y$
 34 $-z, x+1/4, y+1/4$
 35 $z+1/4, -x, y+1/4$
 36 $z+1/4, x+1/4, -y$
 37 y, x, z
 38 $y, -x+1/4, -z+1/4$
 39 $-y+1/4, x, -z+1/4$
 40 $-y+1/4, -x+1/4, z$
 41 x, z, y
 42 $x, -z+1/4, -y+1/4$
 43 $-x+1/4, z, -y+1/4$
 44 $-x+1/4, -z+1/4, y$
 45 z, y, x
 46 $z, -y+1/4, -x+1/4$
 47 $-z+1/4, y, -x+1/4$
 48 $-z+1/4, -y+1/4, x$
 49 $x, y+1/2, z+1/2$
 50 $x, -y+3/4, -z+3/4$
 51 $-x+1/4, y+1/2, -z+3/4$
 52 $-x+1/4, -y+3/4, z+1/2$
 53 $y, z+1/2, x+1/2$
 54 $y, -z+3/4, -x+3/4$
 55 $-y+1/4, z+1/2, -x+3/4$
 56 $-y+1/4, -z+3/4, x+1/2$
 57 $z, x+1/2, y+1/2$
 58 $z, -x+3/4, -y+3/4$
 59 $-z+1/4, x+1/2, -y+3/4$
 60 $-z+1/4, -x+3/4, y+1/2$
 61 $-y, -x+1/2, -z+1/2$
 62 $-y, x+3/4, z+3/4$
 63 $y+1/4, -x+1/2, z+3/4$
 64 $y+1/4, x+3/4, -z+1/2$
 65 $-x, -z+1/2, -y+1/2$
 66 $-x, z+3/4, y+3/4$
 67 $x+1/4, -z+1/2, y+3/4$
 68 $x+1/4, z+3/4, -y+1/2$
 69 $-z, -y+1/2, -x+1/2$

70 $-z, y+3/4, x+3/4$
 71 $z+1/4, -y+1/2, x+3/4$
 72 $z+1/4, y+3/4, -x+1/2$
 73 $-x, -y+1/2, -z+1/2$
 74 $-x, y+3/4, z+3/4$
 75 $x+1/4, -y+1/2, z+3/4$
 76 $x+1/4, y+3/4, -z+1/2$
 77 $-y, -z+1/2, -x+1/2$
 78 $-y, z+3/4, x+3/4$
 79 $y+1/4, -z+1/2, x+3/4$
 80 $y+1/4, z+3/4, -x+1/2$
 81 $-z, -x+1/2, -y+1/2$
 82 $-z, x+3/4, y+3/4$
 83 $z+1/4, -x+1/2, y+3/4$
 84 $z+1/4, x+3/4, -y+1/2$
 85 $y, x+1/2, z+1/2$
 86 $y, -x+3/4, -z+3/4$
 87 $-y+1/4, x+1/2, -z+3/4$
 88 $-y+1/4, -x+3/4, z+1/2$
 89 $x, z+1/2, y+1/2$
 90 $x, -z+3/4, -y+3/4$
 91 $-x+1/4, z+1/2, -y+3/4$
 92 $-x+1/4, -z+3/4, y+1/2$
 93 $z, y+1/2, x+1/2$
 94 $z, -y+3/4, -x+3/4$
 95 $-z+1/4, y+1/2, -x+3/4$
 96 $-z+1/4, -y+3/4, x+1/2$
 97 $x+1/2, y, z+1/2$
 98 $x+1/2, -y+1/4, -z+3/4$
 99 $-x+3/4, y, -z+3/4$
 100 $-x+3/4, -y+1/4, z+1/2$
 101 $y+1/2, z, x+1/2$
 102 $y+1/2, -z+1/4, -x+3/4$
 103 $-y+3/4, z, -x+3/4$
 104 $-y+3/4, -z+1/4, x+1/2$
 105 $z+1/2, x, y+1/2$
 106 $z+1/2, -x+1/4, -y+3/4$
 107 $-z+3/4, x, -y+3/4$
 108 $-z+3/4, -x+1/4, y+1/2$
 109 $-y+1/2, -x, -z+1/2$
 110 $-y+1/2, x+1/4, z+3/4$
 111 $y+3/4, -x, z+3/4$
 112 $y+3/4, x+1/4, -z+1/2$
 113 $-x+1/2, -z, -y+1/2$
 114 $-x+1/2, z+1/4, y+3/4$

115 $x+3/4, -z, y+3/4$
 116 $x+3/4, z+1/4, -y+1/2$
 117 $-z+1/2, -y, -x+1/2$
 118 $-z+1/2, y+1/4, x+3/4$
 119 $z+3/4, -y, x+3/4$
 120 $z+3/4, y+1/4, -x+1/2$
 121 $-x+1/2, -y, -z+1/2$
 122 $-x+1/2, y+1/4, z+3/4$
 123 $x+3/4, -y, z+3/4$
 124 $x+3/4, y+1/4, -z+1/2$
 125 $-y+1/2, -z, -x+1/2$
 126 $-y+1/2, z+1/4, x+3/4$
 127 $y+3/4, -z, x+3/4$
 128 $y+3/4, z+1/4, -x+1/2$
 129 $-z+1/2, -x, -y+1/2$
 130 $-z+1/2, x+1/4, y+3/4$
 131 $z+3/4, -x, y+3/4$
 132 $z+3/4, x+1/4, -y+1/2$
 133 $y+1/2, x, z+1/2$
 134 $y+1/2, -x+1/4, -z+3/4$
 135 $-y+3/4, x, -z+3/4$
 136 $-y+3/4, -x+1/4, z+1/2$
 137 $x+1/2, z, y+1/2$
 138 $x+1/2, -z+1/4, -y+3/4$
 139 $-x+3/4, z, -y+3/4$
 140 $-x+3/4, -z+1/4, y+1/2$
 141 $z+1/2, y, x+1/2$
 142 $z+1/2, -y+1/4, -x+3/4$
 143 $-z+3/4, y, -x+3/4$
 144 $-z+3/4, -y+1/4, x+1/2$
 145 $x+1/2, y+1/2, z$
 146 $x+1/2, -y+3/4, -z+1/4$
 147 $-x+3/4, y+1/2, -z+1/4$
 148 $-x+3/4, -y+3/4, z$
 149 $y+1/2, z+1/2, x$
 150 $y+1/2, -z+3/4, -x+1/4$
 151 $-y+3/4, z+1/2, -x+1/4$
 152 $-y+3/4, -z+3/4, x$
 153 $z+1/2, x+1/2, y$
 154 $z+1/2, -x+3/4, -y+1/4$
 155 $-z+3/4, x+1/2, -y+1/4$
 156 $-z+3/4, -x+3/4, y$
 157 $-y+1/2, -x+1/2, -z$
 158 $-y+1/2, x+3/4, z+1/4$
 159 $y+3/4, -x+1/2, z+1/4$

```

160 y+3/4,x+3/4,-z
161 -x+1/2,-z+1/2,-y
162 -x+1/2,z+3/4,y+1/4
163 x+3/4,-z+1/2,y+1/4
164 x+3/4,z+3/4,-y
165 -z+1/2,-y+1/2,-x
166 -z+1/2,y+3/4,x+1/4
167 z+3/4,-y+1/2,x+1/4
168 z+3/4,y+3/4,-x
169 -x+1/2,-y+1/2,-z
170 -x+1/2,y+3/4,z+1/4
171 x+3/4,-y+1/2,z+1/4
172 x+3/4,y+3/4,-z
173 -y+1/2,-z+1/2,-x
174 -y+1/2,z+3/4,x+1/4
175 y+3/4,-z+1/2,x+1/4
176 y+3/4,z+3/4,-x
177 -z+1/2,-x+1/2,-y
178 -z+1/2,x+3/4,y+1/4
179 z+3/4,-x+1/2,y+1/4
180 z+3/4,x+3/4,-y
181 y+1/2,x+1/2,z
182 y+1/2,-x+3/4,-z+1/4
183 -y+3/4,x+1/2,-z+1/4
184 -y+3/4,-x+3/4,z
185 x+1/2,z+1/2,y
186 x+1/2,-z+3/4,-y+1/4
187 -x+3/4,z+1/2,-y+1/4
188 -x+3/4,-z+3/4,y
189 z+1/2,y+1/2,x
190 z+1/2,-y+3/4,-x+1/4
191 -z+3/4,y+1/2,-x+1/4
192 -z+3/4,-y+3/4,x

```

```

loop_
_atom_site_label
_atom_site_type_symbol
_atom_site_symmetry_multiplicity
_atom_site_Wyckoff_label
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_occupancy
C1 C      8 b 0.37500 0.37500 0.37500 1.00000

```